University of Tartu

Faculty of Science and Technology

Institute of Mathematics and Statistics

Hele-Liis Peedosk

Forecasting time series with artificial neural networks

Actuarial and Financial Engineering

Master's thesis (30 ECTS)

Supervisor: Toomas Raus, PhD

Tartu 2019

Forecasting time series with artificial neural networks

Having accurate time series forecasts helps to be prepared for upcoming events. As many real world time series have nonlinear and irregular behavior, traditional approaches may be lacking performance. A suitable alternative method is artificial neural network models, that can achieve high accuracy in various difficult tasks. The objective of given thesis is to give theoretical and practical guidelines for applying neural networks in time series forecasting with packages h2o and neuralnet for statistical programming language `R`, and library `Keras` for programming language `Python`. An empirical study was conducted on five different datasets to compare multilayer perceptron model performance with long short-term memory model, and iterative, direct and multi-neural network modeling strategies with each other. The performance of neural network models were compared with liner baseline models to expose whether the results have any practical gain. When comparing the network structures, the results indicate the superiority of long short-term memory models. Furthermore, long short-term memory models offered improvement over linear baseline model almost in case of all datasets. Based on these results, neural networks proved to have great performance for time series forecasting, and should be considered as an alternative to linear models.

Keywords: Artificial neural networks, time series analysis, forecasting
CERCS research specialization: Artificial Intelligence (P176)

Aegridade prognoosimine tehisnärvivõrkude meetodil

Aegridade võimalikult täpne prognoosimine võimaldab olla valmis tulevasteks muutusteks. Tihti iseloomustab päriselulisi andmeid mittelineaarsed ja ebaregulaarsed muutused, mida on klassikalistel lineaarsetel aegrea prognoosimudelitel keeruline arvesse võtta. Sellistel juhtudel võib sobilikuks osutuda tehisnärvivõrkude meetod, mis suudab ka keerulistes prognoosiülesannetes hea täpsuse saavutada. Antud töö eesmärgiks on anda nii teoreetilisi kui ka praktilisi juhised tehisnärvivõrkude kasutamiseks aegridade prognoosimisel, kui kasutada programmeerimiskeele `R` jaoks välja töötatud pakette `h2o` ja `neuralnet` ning `Python` jaoks arendatud paketti `Keras`. Viie erineva andmestiku korral viidi läbi protsess, mille käigus võrreldi mitmekihilise närvivõrgu ja rekurrentse närvivõrgu ühe erivormi, LSTMi (ingl.k. *long short-term memory*), tulemusi. Lisaks võrreldi omavahel ka erinevaid prognooside modelleerimise strateegiaid: iteratiivset, otsest kui ka mitme närvivõrgu meetodit. Saadud prognooside headust hinnati võrreldes parima lineaarse (naiivne või ARIMA) baasmudeliga. Parimaks närvivõrkude struktuuriks, mis töös vaadeldud andmestike korral saavutas prognoosimisel parimad tulemused nii võrreldes teiste närvivõrkude kui ka lineaarse baasmudeliga, oli LSTM-mudel. Selle meetodi tulemuste põhjal võib väita, et närvivõrkude kasutamine võib olla heaks alternatiiviks lineaarsetele mudelitele.

Märksõnad: Neurovõrgud, aegridade analüüs, prognostika
CERCS teaduseriala: Tehisintellekt (P176)

Contents

Introduction

Time series forecasting is useful in large variety of domains, such as finance, economics and biology, for example. Forecasting can help to be prepared for what is coming in the future, or even profit from the upcoming changes. In order to have highest benefit, it is needed to obtain forecasts with high accuracy.

Unfortunately, many real world time series are very difficult to forecast. Although there have been many statistical modeling techniques developed, the performance could always be improved. Many of the statistical techniques produce accurate forecasts in case of data with linear relations. The problem arises with nonlinearities that are often difficult to capture. For example, series may contain chaotic component caused by psychological, or even political factors.

In recent decades, a surge of new machine learning method, artificial neural networks, have achieved excellent results in different problems related to computer vision and natural language processing. An artificial neural network is a system of computational units that send information to each other over weighted connections. The parameters of the model are the connecting weights, and these are estimated during training process. The training algorithm seeks to minimize the error of predictions by changing the parameter values in a way that characterizes the population of training sample. The most useful property of artificial neural networks, that could give advantage in time series forecasting, is the ability to capture nonlinearities, and therefore offer a solution where statistical linear methods would have poor performance.

The first objective of this thesis is to give guidance for applying artificial neural networks in time series forecasting tasks. The second objective is to experiment developing univariate neural network models with packages `neuralnet` and `h2o` for `R`, and `Keras` for `Python`, and to compare results in the setting of time series forecasting. The comparison will be based on five datasets that exhibit different nonlinearities. The performance will be evaluated over different forecasting horizons, the number of preceding observations, different modeling strategies, neural network architectures and network structure. The results will be compared with naïve and ARIMA models as baseline.

The thesis will be structured as follows. Section 1 provides fundamental knowledge of artificial neural network models, different architectures and model training process. Section 2 will give brief overview of time series forecasting problem, and linear methods used as baseline models. Furthermore, the section describes how artificial neural networks are applied in time series forecasting tasks. Finally, Section 3 consists of empirical study with results of comparing different neural networks with baseline models in case of five datasets and three programming packages in `R` and `Python`.

1 Artificial Neural Networks

1.1 Introduction

This section is based on the book by Haykin (2009: pp. 1-9) if not referred otherwise.

Artificial neural networks (ANN), or more commonly referred to as neural networks (NN), are machines which consist of layers of simple processing units, each storing experimental knowledge, and sending the information to each other as signals over weighted connections.

Artificial neural networks can be considered as oversimplified models of biological nervous systems, that are mainly used for producing decisions or actions based on some given or obtained input information. The brain, as the core of the nervous system, receives impulses with information from receptors. After the information is perceived, and the decision made, the impulses will be converted to system output by effectors. These concepts that are illustrated also on Figure 1 can be considered as the core of artificial neural networks systems.

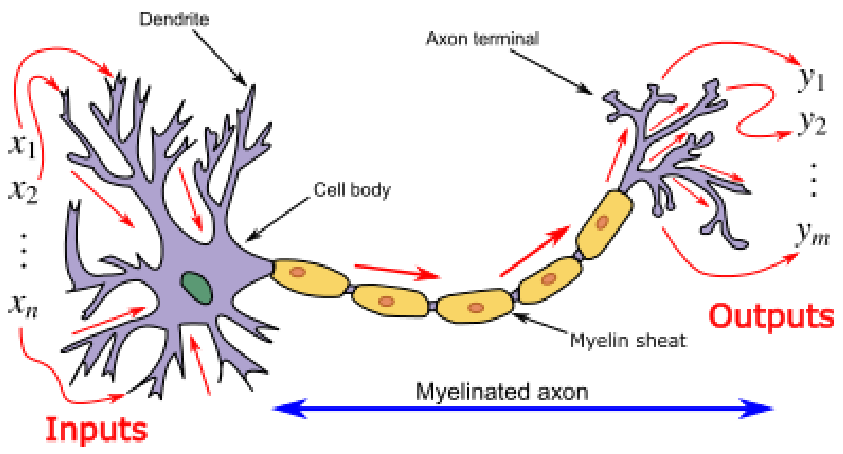


Figure 1. "Nerve cell with signal flows from input as dendritic cell receptors to outputs of effector cells at axon terminals." by Prof. Loc Vu-Quoc is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported licence.

The research towards the artificial neural networks we know today started off from the theory by Hebb (1949) in neuroscience. He claimed that the nerve fibers that communicate persistently with each other will increase the effectiveness of the signals. This idea was taken into use for developing computational methods for pattern recognition, simulating the action between neurons in a biological brain. Rosenblatt (1959) proposed the original Perceptron, one of the first artificial neural network models. Additionally, he provided a proof for a theorem about the convergence of weights of the connections in Perceptron learning algorithm.

Minsky and Papert (1969) published research showing the weakness of perceptron models. They offered a multilayer solution for tackling some drawbacks, but due to the lack of theoretical results, the research for neural networks stagnated.

The early eighties brought extensive research back to life, with newly attained important theoretical results of learning process and improvements in computer processing capacities. A new movement of parallel distributed processes (Rumelhart et al., 1986) based on the idea that when many simple processing units catching smaller distributed pieces of information are combined together, they can achieve intelligent output. This period of extensive research provided many concepts that are still in use.

Another generation of ANN-s started from the work of Hinton (2006), when he managed to show a greedy learning algorithm in case of multiple hidden layers. This result made it possible to train much deeper networks than had been trained before, and has consequently given a competitive edge when compared to other machine learning methods.

There are numerous different neural network architectures developed, and will be developed hereinafter, for various specific tasks of simple classification, regression, image, audio and video processing, natural language processing, etc.

Artificial neural network methods have many very good properties. Firstly, neural networks can capture nonlinear relations. Secondly, the algorithms are computationally powerful as many calculations are run parallel on distributed structure. Thirdly, the models have the ability to generalize the gained knowledge to generate output for objects that had not been seen in the training process. Fourth, networks have capability of adapting interneuron connection weights when there are changes in the environment.

1.2 Fundamentals

This section is based on the book by Haykin (2009: pp. 10-15) if not referred otherwise.

Neural networks consist of information-processing units called neurons, often referred to as nodes. A simple model of a neuron consists of three basic elements: connecting links, a summation and an activation function (Figure 2).

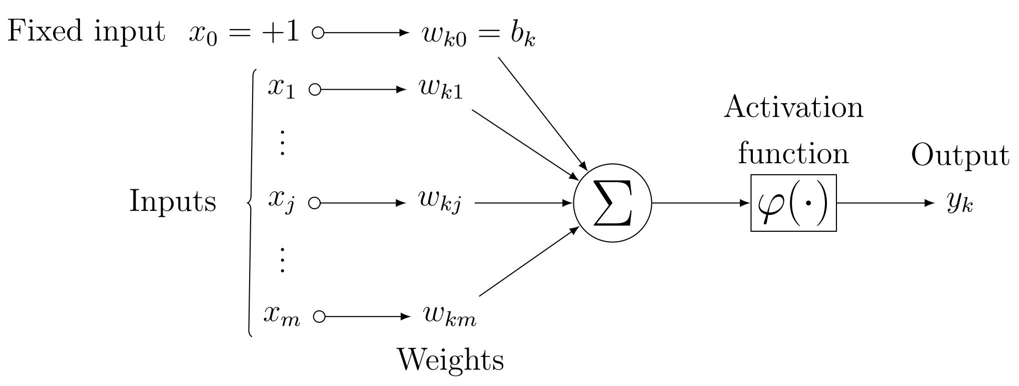


Figure 2. Model of an artificial neuron k.

Each neuron k will get input signals $x_1, \ldots, x_m$. Connecting link, or synapse, connects the input signal x_j with neuron k, and this interneuron link is characterized by the weight w_{kj}. The weights show the strength and importance of given connection, and therefore can take both positive and negative values.

Usually an additional external input called bias b_k is added to the input of a neuron k. It has the ability to increase or decrease the net input of the neuron. Equivalently, it is possible to decompose the bias into external fixed input $x_0 = +1$ with corresponding weight w_{k0}, i.e $b_k = w_{k0}x_0 = w_{k0}$.

All the weighted inputs are summed together into net input v_k. The sum can be expressed as follows:

$$\begin{aligned} v_k &= \sum_{j=1}^{m} w_{kj}x_j + b_k \\ &= \sum_{j=0}^{m} w_{kj}x_j. \end{aligned}$$

The net input of a neuron will be fed into activation function $\varphi(\cdot)$ in order to limit the amplitude of the output y_k in some needed extent. The choice of the activation function depends on task that needs to be solved, and an overview of the most used functions is given below (Goodfellow, Bengio & Courville, 2016: pp. 187-191).

- In regular binary classification task the output should give enough information to separate the classes. Typical activation function in this case is a logistic sigmoid function that squashes the output to interval (0,1) :

  $$\varphi(v_k) = \frac{1}{1 + \exp(-v_k)}.$$

  Sometimes having the output in interval of $(-1,1)$ can give computational advantage, and in this case a suitable function is hyperbolic tangent function:

  $$\varphi(v_k) = \tanh(v_k).$$

- In other cases a very often used activation function for hidden neurons is

  $$\varphi(v_k) = \max\{0, v_k\}.$$

  When this activation function is used then the neuron is called a rectified linear unit. This activation gives great computational advantage as the function has constant derivatives, and this makes the computations in learning algorithms far more efficient.

The output of a single neuron is the result of activation function:

$$y_k = \varphi(v_k) = \varphi\bigg(\sum_{j=0}^{m} w_{kj} x_j\bigg).$$

1.3 Network Architectures

The neural networks can have fundamentally different architectures, and the choice of the architecture will affect the choice of learning algorithms. Haykin (2009, pp. 21-23) covers three classes of them. The first two are strictly layered networks, meaning that the neurons are organized as layers.

Single-Layer Feedforward Networks

This is the simplest layered network, and it consists of two layers - input and output layer. As there are no calculations performed in the input layer, it is not counted and that is why this type of model is called single-layer network. The input nodes are fed directly into output layer, and after some calculations the output of the whole network is produced.

The term feedforward indicates that the information flows through the connecting links only in one direction - from input nodes into output nodes.

Multilayer Feedforward Networks

When there is one or more hidden layers of neurons between input and output layers, then model is referred to as multilayer feedforward network. This architecture also requires the signals to flow only in forward direction - from input layer to first hidden layer, then in case of additional hidden layers consecutively into each of them, and from the last hidden layer into the output layer.

On Figure 3 we can see two-layer network that has 5 input nodes, one hidden layer with 3 computational nodes and an output layer with a single neuron.

By far the most common multilayer feedforward network architecture is multilayer perceptron, which is introduced in detail in Section 1.4. Often the terms *feedforward* *network* and *multilayer perceptron* are used as synonyms (Goodfellow et al., 2016: pp. 164). Another class of models that is also considered as a multilayer feedforward network is convolutional neural networks. This subclass is not introduced in given thesis, but a great overview can be found in Chapter 4.17 of the book by Haykin (2009).

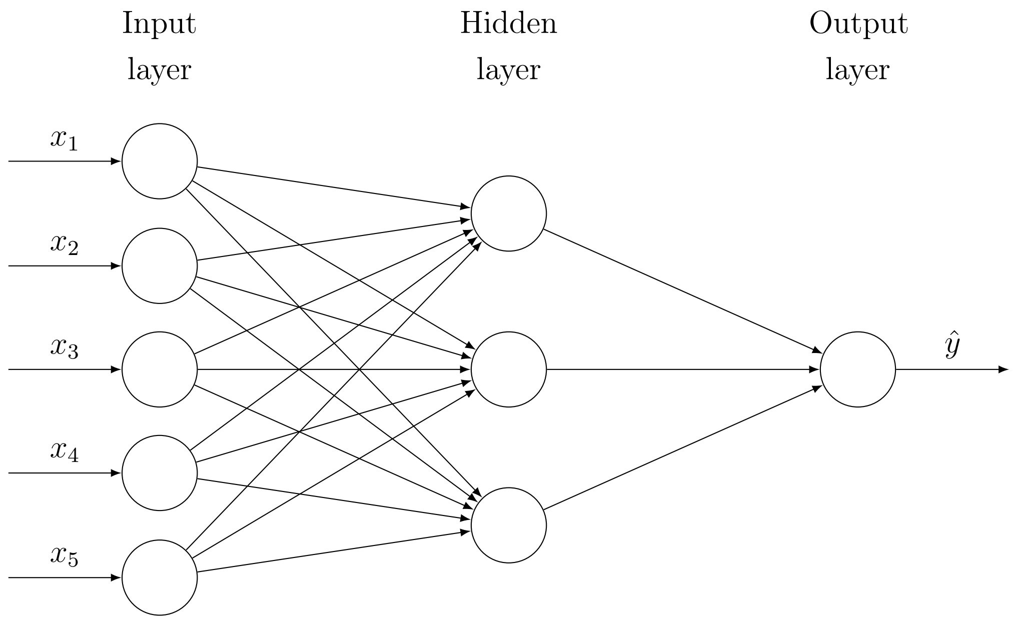


Figure 3. Example of two-layer feedforward network, where $\hat{y}$ denotes the output produced by output neuron is considered as the prediction of model. Based on the number of neurons it can be referred as 5-3-1 network.

Recurrent Networks

The networks where the information flow direction is not strictly one-way, and at least one feedback loop exists, are referred to as recurrent neural networks. The feedback loop allows the output of a neuron to influence the input of itself or any other neuron in the same layer or preceding layer. In the first case, when the output is fed into the same neuron's input, the loops are often called self-feedback loops. There is a feedforward network depicted in Figure 3, but with additional feedback loops it is considered as a recurrent networks (see Figure 4).

The class of recurrent neural networks is very diverse, many special models cast aside the traditional layered structure. A subclass of recurrent neural networks called long short-term memory models are introduced in Section 1.5.2.

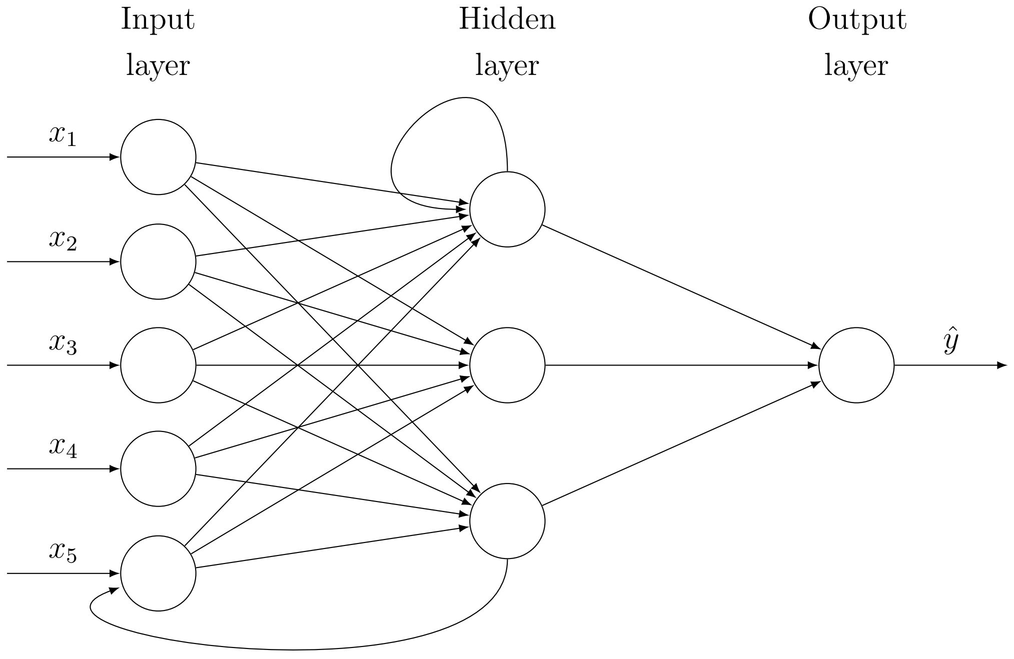


Figure 4. Example of a recurrent neural network, where $\hat{y}$ denotes the prediction for true value y produced by output neuron. The network is recurrent as the first neuron of the hidden layer has a self-feedback loop, and the last neuron of the hidden layer has feedback loop to the last node of input layer.

1.4 Multilayer Perceptron

The sections about multilayer perceptron are based on the book by Haykin (2009: pp. 120-126) if not referred otherwise.

A multilayer perceptron (MLP) is a neural network architecture which contains at least one hidden layer of neurons, and where all the connections between layers are strictly feedforward. A classical multilayer perceptron is fully connected, i.e every neuron in preceding layer is connected to all neurons in the subsequent layer.

Multilayer perceptron can be referred to as *deep* feedforward network (Goodfellow et al., 2016: pp. 164). This term was adapted in late 2000s with the significant improvement in computational power, which made it possible to train deeper networks, remarkably higher number of hidden layers and neurons (Boehmke, 2018).

The role of hidden neurons is to detect different patterns and features that characterize the data. The patterns are discovered as a part of learning process when the input data is transformed into a new feature space. These transformed values attempt to separate the values more easily according to some pattern when comparing with original, untransformed values.

In order to have an effective network, it is needed to find optimal values of free parameters – weights and biases – that identify patterns that characterize the data.

During the learning process the inputs are fed through the network with initial set of weights until the output values are achieved. The outputs are compared with real values, and an error metric will be calculated. The error is then fed into a chosen cost function. Multiple possible cost functions, or often called loss functions, are introduced in Section 1.4.3. The technical objective of the learning process is to minimize the cost function. For this purpose the parameters are modified iteratively after every batch of samples, until a minimum of the cost function is reached. Classical batch consists of all training samples.

The output error raises the issue of credit-assignment problem, i.e how to divide "the blame" of high output error between all the parameters that were used to calculate certain outputs, and how to penalize or reward the hidden neurons in respect to their responsibility. The sizes of the effect on the cost function can be calculated as partial derivatives of the cost function with respect to different parameters, i.e as a gradient vector of the cost function. The gradient will be calculated based on the back-propagation algorithm introduced in Section 1.4.1. The parameters are then penalized or rewarded in accordance to the gradient with opposite sign, in order to reduce the function. This method is called gradient decent method, and will be together with an improved algorithm of it, called as stochastic gradient descent method, introduced in Section 1.4.2.

1.4.1 The Back-Propagation Algorithm

The notation in this subsection is following the book by Haykin (2009: pp. 129-141), and it is integrated with technical nuances by Goodfellow et al. (2016: pp. 200-209).

The algorithm consists of two parts – the forward propagation and the backward propagation. In the first part, an observation, or set of observations, are fed through the network, and the corresponding cost values are computed. During the backward propagation step, the gradients for parameters - weights and biases - are computed.

The complexity of the algorithm increases with the number of hidden layers. First, the gradients are computed for output layer. Second, the gradients will be computed subsequently for each preceding hidden layer. Derivation of gradient vectors is described for both types of layers in the following paragraphs. Additionally, the full algorithm for fully connected multilayer perceptron that optimizes run-time by keeping the values of reoccurring computations in memory is presented as Algorithm 1 in Appendix A.

Deriving the Gradient Vector of the Cost Function for Output Layer

Consider n-th iteration of learning process. Let the output layer be the l-th layer in the network, let the output layer consist of m_l neurons. In the notation for output value $y_j^{(l)}(n)$ the superscript l indicates the layer, subscript j indicates the neuron j in the corresponding layer, and argument n indicates the iteration step. As $y_j^{(l)}(n)$ is output value on the output layer, and therefore considered as prediction for the observation, the output will be denoted with hat, i.e, $\hat{y}_j^{(l)}(n)$, to separate true values and predictions of given observation.

Recall, the output $\hat{y}_j^{(l)}(n)$ of a neuron $j \in \{1, 2, \ldots, m_l\}$ in the l-th layer can be calculated as a result of an activation function over net input for the neuron j, i.e

$$\hat{y}_j^{(l)}(n) = \varphi_j(v_j^{(l)}(n)) = \varphi_j\left(\sum_{i=0}^{m_{l-1}} w_{ji}^{(l)}(n) y_i^{(l-1)}(n)\right),$$

where $y_i^{(l-1)}(n), i = 1, \ldots m_{l-1}$ are outputs from preceding $(l-1)$-th hidden layer with m_{l-1} neurons as input for neuron j, $y_0^{(l-1)}(n)$ is the external input fixed to size $+1$, $w_{ji}^{(l)}(n), i = 0, \ldots, m$ are the weights of the connecting links from input $y_i^{(l-1)}(n)$ to neuron j, and $\varphi(\cdot)$ is the activation function. For simplicity the notation is abbreviated by leaving out the iteration step argument (n) as all the calculations presented in this section are for the iteration step n.

In the matrix notation the last formula can be expressed as a dot product of vectors $\boldsymbol{W_j^{(l)}} = \left(w_{j1}^{(l)}, w_{j2}^{(l)}, \ldots, w_{jm_{l-1}}^{(l)}\right)$ and $\boldsymbol{y^{(l-1)}} = \left(y_1^{(l-1)}, y_2^{(l-1)}, \ldots, y_{m_{l-1}}^{(l-1)}\right)^T$ in the following way

$$\hat{y}_j^{(l)} = \varphi_j\big(\boldsymbol{W_j^{(l)}} \boldsymbol{y^{(l-1)}} + b_j\big).$$

The error produced by output $\hat{y}_j^{(l)}$ is calculated as a difference of output value and the true value y_j, and this can be expressed as

$$e_j = y_j - \hat{y}_j^{(l)}.$$

The cost function $\mathcal{E}(\cdot)$ is a continuously differentiable function of the parameters $\boldsymbol{W_j^{(l)}}$. To find the gradient, it must be noted that $\mathcal{E}(\boldsymbol{W_j^{(l)}})$ is a function composition, and therefore the chain rule of calculus must be used.

The gradient of a cost function on the weights on neuron j in the output layer $\nabla_{\boldsymbol{W_j^{(l)}}} \mathcal{E}$ is expressed as a vector in the following way

$$\nabla_{\boldsymbol{W_j^{(l)}}} \mathcal{E} = \left(\frac{\partial \mathcal{E}}{\partial w_{j1}^{(l)}}, \frac{\partial \mathcal{E}}{\partial w_{j2}^{(l)}}, \ldots, \frac{\partial \mathcal{E}}{\partial w_{jm_{l-1}}^{(l)}}\right).$$

Each partial derivative can be calculated according to chain rule as

$$\frac{\partial \mathcal{E}}{\partial w_{ji}^{(l)}} = \frac{\partial \mathcal{E}}{\partial e_j} \frac{\partial e_j}{\partial \hat{y}_j^{(l)}} \frac{\partial y_j^{(l)}}{\partial v_j^{(l)}} \frac{\partial v_j^{(l)}}{\partial w_{ji}^{(l)}}.$$

Taking the derivatives with respect to $\hat{y}_j^{(l)}$ on the both sides of the equation $e_j = y_j - \hat{y}_j^{(l)}$ yields

$$\frac{\partial e_j}{\partial \hat{y}_j^{(l)}} = -1.$$

Taking the derivatives with respect to $v_j^{(l)}$ on the both sides of the equation $\hat{y}_j^{(l)} = \varphi_j(v_j^{(l)})$ yields

$$\frac{\partial \hat{y}_j^{(l)}}{\partial v_j^{(l)}} = \varphi_j'(v_j^{(l)}).$$

Taking the derivatives with respect to $w_{ji}^{(l)}$ on the both sides of the equation $v_j^{(l)} = \sum_{i=0}^{m_{l-1}} w_{ji}^{(l)} y_i^{(l-1)}$ yields

$$\frac{\partial v_j^{(l)}}{\partial w_{ji}^{(l)}} = y_i^{(l-1)}.$$

Therefore the partial derivative $\frac{\partial \mathcal{E}}{\partial w_{ji}^{(l)}}$ is calculated as

$$\frac{\partial \mathcal{E}}{\partial w_{ji}^{(l)}} = -\frac{\partial \mathcal{E}}{\partial e_j^{(l)}} \cdot \varphi_j'(v_j^{(l)}) y_i^{(l-1)}.$$

The gradient matrix on all weights is as follows

$$\nabla_{\boldsymbol{W}^{(l)}} \mathcal{E} = \left(\nabla_{\boldsymbol{W}_1^{(l)}} \mathcal{E}, \ldots, \nabla_{\boldsymbol{W}_{m_l}^{(l)}} \mathcal{E} \right).$$

Finding the Gradient Vector of the Cost Function for Hidden Layers

When the neuron j is in the output layer l, the gradient calculations are as straightforward as shown above, because the error signal is computed when comparing with the true value. But when the neuron j is in a hidden layer, then there is no true output value to compute exact error, and therefore the error signal must be fed backwards from the output neuron k of output layer l to preceding hidden layers. Figure 5 depicts the signal-flow between neuron j in the last, $(l-1)$-th hidden layer and the error e_k produced by the output of neuron k in the output layer.

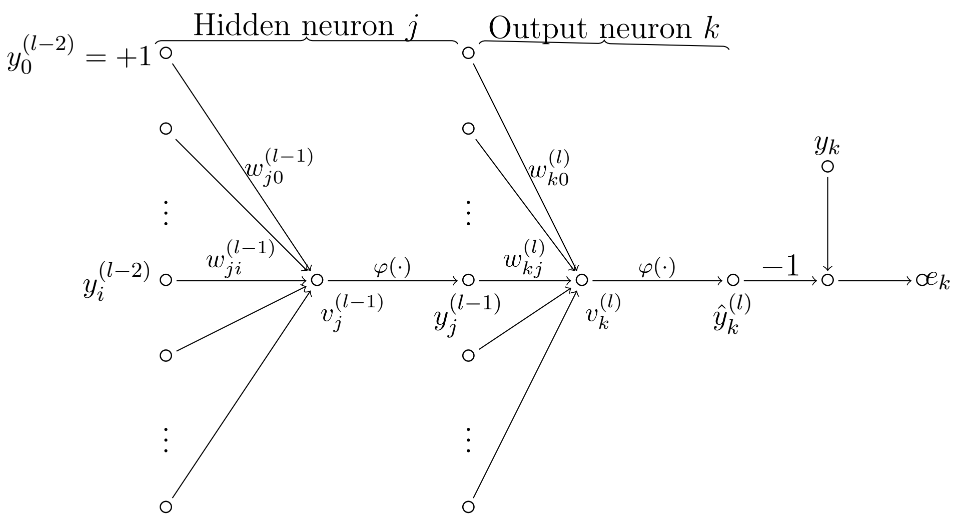


Figure 5. Signal-flow graph illustrating how the output neuron k is connected with neuron j in preceding hidden layer. (Haykin, 2009: pp. 132)

The gradient vector of weights $\nabla_{\boldsymbol{W}_j^{(l-1)}}\mathcal{E}$ of neuron j in (l-1)-th layer consists in this case of partial derivatives $\frac{\partial \mathcal{E}}{\partial w_{ji}^{(l-1)}}$ that are derived in the similar way as for the output neurons shown above,

$$\frac{\partial \mathcal{E}}{\partial w_{ji}^{(l-1)}} = \frac{\partial \mathcal{E}}{\partial y_j^{(l-1)}} \frac{\partial y_j^{(l-1)}}{\partial v_j^{(l-1)}} \frac{\partial v_j^{(l-1)}}{\partial w_{ji}^{(l-1)}}.$$

The main difference is that the partial derivative $\frac{\partial \mathcal{E}}{\partial y_j^{(l-1)}}$ is a sum over all output layer neurons error effects on output $y_j^{(l)}$

$$\frac{\partial \mathcal{E}}{\partial y_j^{(l-1)}} = \sum_{k=1}^{m_l} \frac{\partial \mathcal{E}}{\partial e_k} \frac{\partial e_k}{\partial \hat{y}_k^{(l)}} = \sum_{k=1}^{m_l} \frac{\partial \mathcal{E}}{\partial e_k} \frac{\partial e_k}{\partial v_k^{(l)}} \frac{\partial v_k^{(l)}}{\partial y_j^{(l-1)}}.$$

The partial derivative $\frac{\partial e_k}{\partial v_k^{(l)}}$ can be calculated by taking derivative of $e_k = y_k - \hat{y}_k^{(l)} = y_k - \varphi_k(v_k^{(l)})$ with respect to $v_k^{(l)}$ as follows

$$\frac{\partial e_k}{\partial v_k^{(l)}} = -\varphi_k'(v_k^{(l)}).$$

The net input $v_k^{(l)}$ of neuron k is based on the outputs $y_j^{(l-1)}$ of the preceding hidden layer and an external input $y_0^{(l-1)} = +1$, i.e

$$v_k^{(l)} = \sum_{j=0}^{m_{l-1}} w_{kj}^{(l)} y_j^{(l-1)}.$$

The partial derivative of $\frac{\partial v_k^{(l)}}{\partial y_j^{(l-1)}}$ is equal to

$$\frac{\partial v_k^{(l)}}{\partial y_j^{(l-1)}} = w_{kj}^{(l)}.$$

By using the expressions derived above the desired partial derivative $\frac{\partial \mathcal{E}}{\partial w_{ji}^{(l-1)}}$ can be calculated as

$$
\begin{aligned}
\frac{\partial \mathcal{E}}{\partial w_{ji}^{(l-1)}} =& \left(\sum_{k=1}^{m_l} \frac{\partial \mathcal{E}}{\partial e_k} \frac{\partial e_k}{\partial v_k^{(l)}} \frac{\partial v_k^{(l)}}{\partial y_j^{(l-1)}} \right) \frac{\partial y_j^{(l-1)}}{\partial v_j^{(l-1)}} \frac{\partial v_j^{(l-1)}}{\partial w_{ji}^{(l-1)}} = \\
=& \left(- \sum_{k=1}^{m_l} \frac{\partial \mathcal{E}}{\partial e_k} \cdot \varphi_k'(v_k^{(l)}) w_{kj}^{(l)} \right) \cdot \varphi_j'(v_j^{(l-1)}) y_i^{(l-2)}.
\end{aligned}
$$

1.4.2 Stochastic Gradient Descent Method

This section is based on the book by Goodfellow et al. (2016: pp. 80-84, 149-150, 290-296).

Stochastic gradient decent (SGD) is one of the most used algorithms for training a deep learning model. It is based on the gradient decent algorithm, but introduces an accelerated approach to finish the learning process. Therefore, the first following paragraph will describe the gradient decent method to introduce the general approach, and the subsequent paragraph will add a feature to general approach in order to accelerate the learning process.

The Gradient Descent Algorithm

The gradient descent method is the technique of reducing a value of a function $f(\cdot)$, e.g in the learning process a cost function $\mathcal{E}(\cdot)$, in the direction of the opposite sign of the derivative. We know that the value $f(x - \eta \cdot \text{sign}(f'(x))) < f(x)$, when η is small.

The gradient decent method uses entire training set at every iteration, and changes the parameters in order to minimize the cost.

The back-propagation algorithm yields a gradient for cost function, and therefore changing the weights in the direction of negative gradient will reduce to cost. The adjusted matrix of weights $\mathbf{W}^{(k)*}$ that will reduce the cost, can be calculated in the following way

$$
\begin{aligned}
\mathbf{W}^{(k)*} &\leftarrow \mathbf{W}^{(k)} - \eta \nabla_{\mathbf{W}^{(k)}} \mathcal{E}(\hat{\mathbf{y}}^{(k)}, \mathbf{y}), \\
\mathbf{b}^{(k)*} &\leftarrow \mathbf{b}^{(k)} - \eta \nabla_{\mathbf{b}^{(k)}} \mathcal{E}(\hat{\mathbf{y}}^{(k)}, \mathbf{y}),
\end{aligned}
$$

where k iterates over layers, i.e., $k \in \{1,2,\ldots,l\}$, and where a positive scalar η is called the learning rate.

The learning rate is the parameter with the strongest impact on the convergence speed of gradient descent. One possibility for the choice of learning rate is to use a fixed value η. Using too small value of η may result in a slow convergence, but too large value can make the learning process unstable. An additional option is to fix a schedule of $\eta_1, \eta_2, \ldots,$ where the learning rates gradually decrease over time. It is sufficient to choose the schedule based on two conditions to guarantee the convergence of the method. The conditions are following,

$$\sum_{i=1}^{\infty} \eta_i = \infty, \text{and}$$
$$\sum_{i=1}^{\infty} \eta_i^2 < \infty.$$

An another approach to optimize the learning process is to add momentum term. The term is composed as a weighted sum of past gradients, and this can act as accelerator or stabilizing term for the calculated gradient. It can reduce the effect of computed gradient when the sign of momentum term and computed gradient do not concur, or in the opposite it can magnify the effect of the gradient when the signs concur.

When using the momentum terms $\mathbf{m}_W, \mathbf{m}_b$ the parameters are updated in the following way

$$\begin{aligned}
\mathbf{m}_W &\leftarrow \alpha \mathbf{m}_W - \eta \nabla_{\mathbf{W}^{(k)}} \mathcal{E}(\hat{\mathbf{y}}^{(k)}, \mathbf{y}), \\
\mathbf{W}^{(k)*} &\leftarrow \mathbf{W}^{(k)} + \mathbf{m}_W, \\
\mathbf{m}_b &\leftarrow \alpha \mathbf{m}_b - \eta \nabla_{\mathbf{b}^{(k)}} \mathcal{E}(\hat{\mathbf{y}}^{(k)}, \mathbf{y}), \\
\mathbf{b}^{(k)*} &\leftarrow \mathbf{b}^{(k)} + \mathbf{m}_b,
\end{aligned}$$

where α is a hyperparameter to change the size of impact of previous gradients, $\alpha \in [0,1)$. With higher α the previously calculated gradients affect the outcome with higher magnitude.

The stopping criterion for gradient decent is the convergence, which has been reached when the gradient is zero-vector. In practice, it is often enough when the gradient has reached a small value close to zero.

The Stochastic Gradient Method

Evaluating the cost at every iteration step over all training observations can be computationally expensive. Furthermore, using larger training set does not improve the estimation of gradient vector by as large factor. This has motivated further development of the regular gradient decent into the stochastic gradient decent algorithm presented as Algorithm 2 in Appendix A.

Instead of using the whole training set for computing the cost in one iteration, only a small sample, also referred to as minibatch, is used. The observations are drawn randomly from the training set, and typically the size of a minibatch ranges from $2^4 = 16$ to $2^8 = 256$. Averaging gradients over the minibatch gives an unbiased estimate for the gradient.

Stochastic gradient decent method gives an important advantage over regular gradient decent as the time used for computation stays on the same level even when the training set is larger. Furthermore, the algorithm can achieve tolerated maximal limit of cost before it has processed all the observations in training set.

1.4.3 Cost Functions

Changhau (2017) has gathered together most of the cost functions, often referred to as loss functions, that are used by deep learning models. The cost function that will be minimized must be chosen according to the problem type - regression or classification, and whether the overestimates or underestimates should be punished. The author brought out following cost functions for training a regression model.

Consider error $e^{(i)} = y^{(i)} - \hat{y}^{(i)}$ of i-th observation in training batch, where $y^{(i)}$ is true value of an output of observation, $\hat{y}^{(i)}$ is the predicted value. Let cost function be $E = \mathcal{E}(\theta)$, where θ denotes all parameters, the weights and biases.

- **L1 Loss** function sums the absolute errors of n observations, and it is calculated as
$$E = \sum_{i=1}^{n} |e^{(i)}|.$$
- **Mean Absolute Error** (MAE) measures the distance between predicted and true values. It is calculated as the L1 loss, but the result of L1 is averaged between observations. Therefore the cost function of MAE is
$$E = \frac{1}{n}\sum_{i=1}^{n} |e^{(i)}|.$$
- **Mean Absolute Percentage Error** (MAPE) measures the percentage error between predicted and true values. It is calculated as
$$E = \frac{1}{n}\sum_{i=1}^{n} \left|\frac{e^{(i)}}{y^{(i)}}\right| \cdot 100.$$
- **L2 Loss** function, often referred to as sum of squared errors, sums the squared errors of n observations, and it is calculated as
$$E = \sum_{i=1}^{n} (e^{(i)})^2.$$

- **Mean Squared Error** (MSE) is calculated as the L2 loss, but the result of L2 is averaged between observations. Therefore the cost function of MSE is

$$E = \frac{1}{n}\sum_{i=1}^{n}(e^{(i)})^2.$$

- **Mean Squared Logarithmic Error** (MSLE) is a variant of MSE where the prediction and true value are log-transformed. The corresponding cost function of MSLE is calculated as

$$E = \frac{1}{n}\sum_{i=1}^{n}\big(\log(y^{(i)}+1) - \log(\hat{y}^{(i)}+1)\big)^2.$$

The choice of a certain cost function is affected by the interpretation of outliers that produce high error. By using the MAE statistic the outliers do not affect the cost function as much as in case of MSE, and therefore MAE is preferable in the case when the outliers occur more often, and may be due to an erroneous measurement. With MSE the huge differences will cause a very high cost, and is therefore preferred when the measurement is done correctly, and the value is truly an outlier in the output space. In contrary, MSLE is able to penalize more the under-estimates than the over-estimates.

1.5 Long Short-Term Memory Model

Recurrent networks conquer the problem of storing previously processed information for using it later for upcoming predictions. Long short-term memory (LSTM) model introduced by Hochreiter & Schmidhuber (1997) is one of the most effective approaches of recurrent networks. A brief overview is based on essay by Olah (2015).

Firstly, the general concept of unrolled recurrent neural networks is introduced in Section 1.5.1. Secondly, in Section 1.5.2 the idea of unrolled recurrent neural networks was used in order to describe the structure of long short-term memory models.

1.5.1 Unrolled Recurrent Neural Network

Recurrent neural networks differ from feedforward networks as they have feedback loops. These loops can be unrolled and visualized as sequences, where every module, or cell, in the chain will produce two types of output - the prediction for the output value, and the information passed to next module as input (Figure 6).

This chain-like representation of recurrent neural networks make it suitable for sequence modeling tasks. These tasks include natural language processing, speech recognition, video processing, time series forecasting, etc.

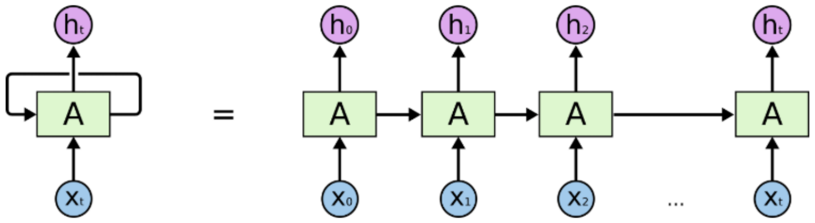


Figure 6. Unrolled recurrent neural network. (Olah, 2015). Retrieved from `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

1.5.2 The Structure of Long Short-Term Memory Models

The LSTM model structure is based on unrolled recurrent neural network depicted on Figure 6. In LSTM model every cell of the chain is constructed in a special way, as shown on Figure 7.

Every LSTM cell consists of three gates with sigmoid layer: forget gate, input gate and output gate, and the modules pass information sequentially through cell states C_t and hidden states h_t, where t is the index of the value in a sequence.

Cell state C_{t-1} contains information from the past that is passed on to a t-th cell. As this past information may contain information that has lost its relevance, the information flow is limited by the forget gate value f_t, which assigns a value in interval [0,1]. If the forget gate value is close to one, most of the past information can continue the flow, but if the value is close to zero, most of the past information is forgotten. The value of forget gate is calculated as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f),$$

where W_f and b_f are corresponding weights matrix and bias vector, x_t is input at t, and $\sigma(\cdot)$ indicates sigmoid function.

The next, input gate, decides which information is needed to pass as information update to cell state. The input gate layer value is calculated as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i).$$

The candidate values for update process, $\tilde{C}_t$, are obtained by squashing input to interval [−1,1] by $\tanh(\cdot)$ function. The values $\tilde{C}_t$ result from

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C).$$

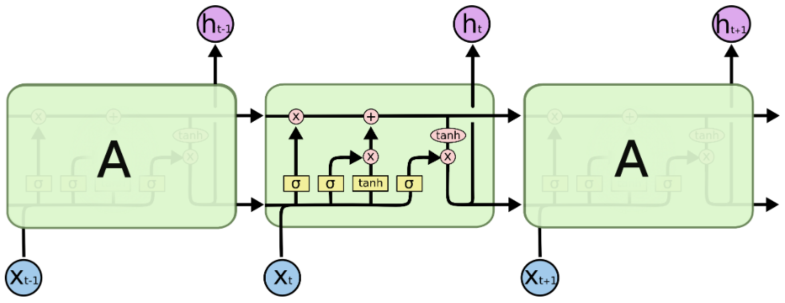


Figure 7. The structure of a LSTM model cell. The upper horizontal line indicates cell state flow, the lower horizontal connections hidden state flow. Yellow boxes represent learned network layers, and pink circles pointwise operations. From "Understanding LSTM Networks" by Olah (2015). Retrieved from `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

The new cell state C_t is a linear combination of remained information from past cell state and new information added, i.e, $C_t = f_t C_{t-1} + i_t \tilde{C}_t$.

The last gate, output gate, will decide which information is going to be the output, i.e, the output gate layer value o_t is filtering values from cell state C_t in order to produce the output. The corresponding values are evaluated in similar way to f_t and i_t,

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o).$$

Before filtering, the cell state value C_t is squashed by $\tanh(\cdot)$ function. The output value h_t, calculated as follows:

$$h_t = o_t \tanh(C_t),$$

is returned as prediction $\hat{y}_t = h_t$ at time step t, and it is also forwarded to the next module at time $t + 1$ as input.

2 Time Series Forecasting

The following two subsections, Section 2.1 and Section 2.2, giving brief overview of time series, and introducing classical forecasting methods are based on book by Brockwell and Davis (2002: pp. 15, 23-24, 29-33, 50, 55, 180, 203) if not noted otherwise.

2.1 Fundamentals

A time series is a set of sequential observations y_t ordered by the time point of measurement t. A time series is referred to as discrete time series when the set of observation time points t is a countable set, $t = 1,2,\dots$. The observations are denoted as y_t.

The forecast of an outcome at unobserved time point, $T + h$, is based on the values measured earlier. Here T is a number of observed values, and h shows which successive time point after T is forecast. The prediction at $T + h$, given on values y_T is denoted as $\hat{y}_{T+h|T}$.

Time series prediction starts usually with plotting the series, and observing it to identify whether the series has trend, seasonal component, apparent behavioral changes, or any outlying observations. In case of trend, and/or seasonal component, it must be taken into account and eliminated, as required by most forecasting methods.

First, the time series can be decomposed to trend T_t, seasonal component S_t, and random error component ε_t, as given in

$$y_t = f(S_t,T_t,\varepsilon_t).$$

The most used formulations of function f are

- $y_t = S_t + T_t + \varepsilon_t$,
- $y_t = S_t \cdot T_t \cdot \varepsilon_t$.

An example of decomposed time series is on Figure 8.

Second, trend and seasonal component can be eliminated by differencing the series. In the following section ∇ operator denotes lag-1 difference computed as

$$\nabla y_t = y_t - y_{t-1} = (1 - B)y_t,$$

where B is operator for backwards shift,

$$By_t = y_{t-1}.$$

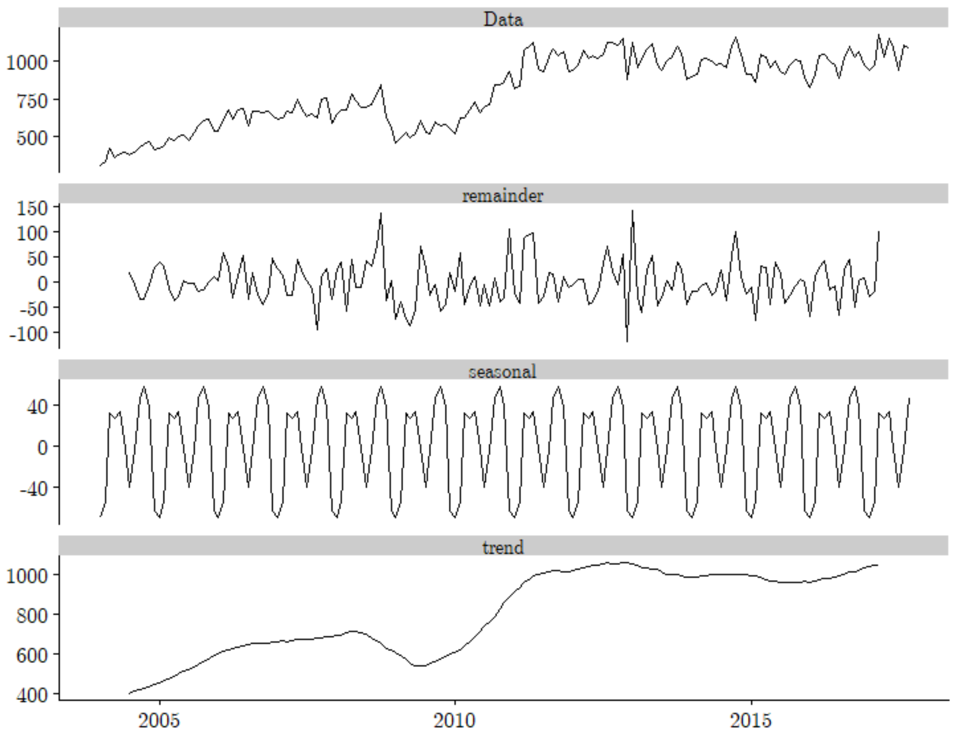


Figure 8. An example of time series additive decomposition.

It is possible to achieve a sequence $\{\nabla^k y_t\}_{t=1}^T = \{\nabla(\nabla^{k-1} y_t)\}_{t=1}^T$ that has no trend. The seasonal component can be eliminated by using seasonal differencing with seasonal differencing operator ∇_s as

$$\nabla_s y_t = y_t - y_{t-s} = (1 - B^s) y_t,$$

where s is the length of one seasonal period (e.g, $s = 12$ for monthly data).

The property that is desired by most methods of time series forecasting is stationarity. A series is considered to be stationary when the statistical properties are similar for every possible sub-series drawn from it. Detrending is the most helpful way to make the series stationary for a lot, not to say for the most of time series.

2.2 Linear methods

2.2.1 Naïve method

The authors Hyndman and Athanasopoulos (2018) have given brief overview of naïve forecasting methods as follows.

Naïve forecast at some successive time point $T + h$ is simply the last observed value of the series, i.e,

$$\hat{y}_{T+h|T} = y_T.$$

The simple naïve forecast is optimal for predicting random walk, therefore this simple method gives good predictions for financial time series.

In case of seasonal component, the naïve forecast is equal to the last observation at the same season, i.e,

$$\hat{y}_{T+h|T} = y_{T+h-s(k+1)},$$

where k is the number of complete seasonal periods for the length of h calculated as $k = \lfloor \frac{h-1}{s} \rfloor$.

2.2.2 ARIMA model

ARIMA stands for autoregressive integrated moving-average models. ARIMA(p,d,q) model is able to handle a series generated by autoregressive process of order p, AR(p), and moving-average process of order q, MA(q), even in case of non-stationary series, if it is possible to make it stationary by differencing it d times.

A series $\{Y_t\}_{t=1}^{T}$ is generated by moving-average process of order q if $\tilde{Y}_t = Y_t - E(Y_t)$ is linear combination of previous q forecast errors, random values, i.e.,

$$\tilde{Y}_t = Z_t + \theta_1 Z_{t-1} + \theta_2 Z_{t-2} + \cdots + \theta_q Z_{t-q},$$

where β is intercept, $Z_t \sim \mathrm{N}(0, \sigma^2)$ are independent random variables, and $\theta_1, \ldots, \theta_q$ are parameters.

A series is generated by autoregressive process of order p if $\tilde{Y}_t$ can be expressed as linear combination of previous p values of the same series, i.e,

$$\tilde{Y}_t = Z_t + \phi_1 \tilde{Y}_{t-1} + \phi_2 \tilde{Y}_{t-2} + \cdots + \phi_p \tilde{Y}_{t-p},$$

where $Z_t \sim \mathrm{N}(0, \sigma^2)$ are independent random variables, and $\phi_1, \ldots, \phi_p$ are parameters.

A series $\{Y_t\}_{t=1}^{T}$ is generated by ARIMA(p,d,q) process if

$$\nabla^d \tilde{Y}_t = Z_t + \sum_{i=1}^{p} \phi_i \nabla^d \tilde{Y}_{t-i} + \sum_{j=1}^{q} \theta_j Z_{t-j},$$

or equally,

$$\phi(B)(1-B)^d \tilde{Y}_t = \theta(B) Z_t,$$

where $\theta(\cdot)$ and $\phi(\cdot)$ are following polynomials:

$$\theta(x) = 1 + \sum_{j=1}^{q} \theta_j x^j$$

and

$$\phi(x) = 1 - \sum_{i=1}^{p} \phi_i x^i.$$

Seasonal ARIMA model

ARIMA is able to take into account seasonal effect by introducing seasonal autoregressive and moving-average processes, and seasonal differencing. The models in this subclass are referred to as SARIMA models.

A series $\{Y_t\}_{t=1}^{T}$ is generated by SARIMA(p,d,q) $\times$ $(P, D, Q)_s$ process if

$$\phi(B)\Phi(B^s)(1-B)^d(1-B^s)^D\tilde{Y}_t = \theta(B)\Theta(B^s)Z_t,$$

where s is period of seasonal component, D is the order of lag-s differencing operator, P and Q are correspondingly the orders of seasonal autoregressive and seasonal moving-average processes, and $\Theta(\cdot)$ and $\Phi(\cdot)$ are following polynomials:

$$\Theta(x) = 1 + \sum_{j=1}^{Q} \Theta_j x^j$$

and

$$\Phi(x) = 1 - \sum_{i=1}^{P} \Phi_i x^i.$$

2.3 Artificial Neural Network Methods

The predictive power of artificial neural networks lies in both, the ability to find different patterns in the data, and in the ability to capture nonlinear relations (Haykin, 2009: pp. 1-2). These are highly desired properties in time series forecasting, and this has driven numerous authors to apply neural networks for time series forecasting problems in various domains, e.g., signal processing (Lapedes & Farber, 1987; Carillo, González & Gracia-Linares, 2015), weather forecast (Maqsood, Khan & Abraham, 2004), economical series of production, consumption (Srinivasan, Liew & Chang, 1994), forecasting financial series of stock prices (White, 1988; Siami-Namin & Siami Namin, 2018) and currency prices (Kuan & Liu, 1995; Adhikari & Agrawal, 2013).

Artificial neural networks methods used in time series methods include variety of different architectures. The simplest architecture used is feedforward network, which includes multilayer perceptron (Carillo et al, 2015), radial basis function models (Montaño Moreno, Pol & Gracia, 2011) and extreme learning machines (Singh & Balasundaram, 2007). Also, recurrent networks are applied, Elman network (Kuan & Liu, 1995), and long short-term memory network (Siami-Namin & Siami Namin, 2018), for example.

Most of the network models use the idea of autoregressive series. The input for the model will be a set of previously observed values, and the output node contains the future values to be forecast ahead (Zhang, Patuwo & Hu, 1998). Additionally, more variables can be given as input, e.g., weekday or month identificator, or related time series. For instance, for forecasting temperature values, it is possible to add as input both, previous temperature and atmospheric pressure values.

Two widely used architectures, multilayer perceptron and long short-term memory model, are used in the comprehensive comparison in given work. Strategies how to apply these architectures in time series forecasting task will be covered in Section 2.3.2.

When forecasting time series H steps ahead it is possible to follow one of the following approaches. It is possible to forecast just the H-th step ahead, but it is also possible to forecast the value for each time step h until H. In this thesis the second approach is followed, as the first can be forecast with similar approaches as the second. Predictions for the full horizon H can be forecast by three different methods introduced in Section 2.3.2, and H-step-ahead forecast can be obtained by two of those.

2.3.1 Data Preprocessing

When training neural networks for any type of predictive task, it is often beneficial to apply preprocessing transformations prior to model training.

One way to improve the optimization algorithms is to standardize the data prior to training process. This will ensure that even when the weights are initialized randomly, the algorithm converges faster, as it does not have to find optimal weights from noticeably different range of values (Bishop, 1995: pp. 296-298). Standardized values of a feature y are computed as

$$\tilde{y_i} = \frac{y_i - \bar{y}}{s},$$

where $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ is sample mean, s is unbiased sample standard deviation calculated as $s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y})^2}$, and N is sample size.

However, it must be noted that parameters mean and variance for standardization must be estimated from the training data. When the trained network is used to forecast based on new samples, the data must be standardized based on parameters estimated from training data.

When using ARIMA model in time series forecasting it is required that the series can be made stationary by differencing. So far there is no consensus which data preprocessing methods should be used prior developing a neural network model as concluded by Makridakis, Spiliotis and Assimakopoulos (2018). The series can be deseasonalized, detrended and/or transformed using Box-Cox power transformations in order to achieve better and more stable forecast performance. The decision should be based on particular data, often as a result of comparing forecasts by models trained with transformed and original data.

2.3.2 Modeling Strategies

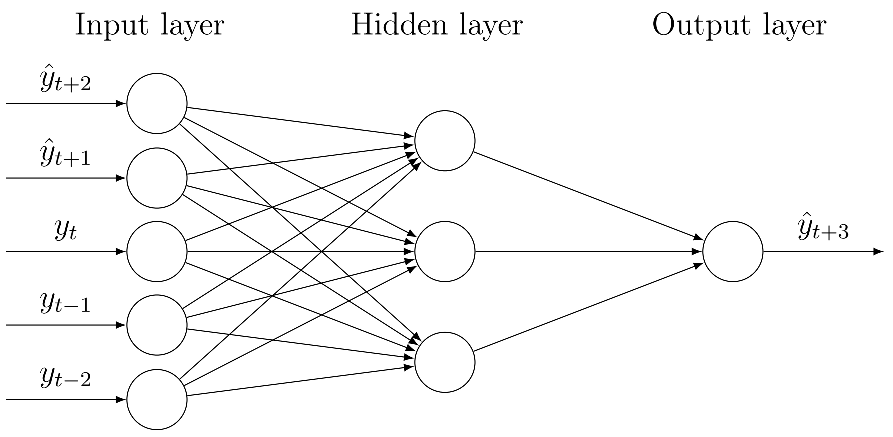
There are three main strategies of using neural networks in forecasting time series. In case of one-step forecasting problem, all the approaches will give same results, but in multi-step forecasting task the forecasts will be calculated differently.

Iterative Strategy
The iterative strategy will produce one model to predict all $h = 1, \dots, H$ steps in the future. This model has only one output neuron that will predict value at next time step, and in order to achieve H forecasts it is iterated H times. The predicted value of one iteration will be used as an input for forecasting the value at next time step. This results in forecasts $\hat{y}_{t+h}$, that are computed given on $n+1$ past values: $\hat{y}_{t+(h-1)}, \dots, \hat{y}_{t+1}, y_t, \dots, y_{t-n+(h-1)}$. An example of the iterative approach with multilayer perceptron architecture in case of $h = 3$ and $n = 4$ is given on Figure 9(a).

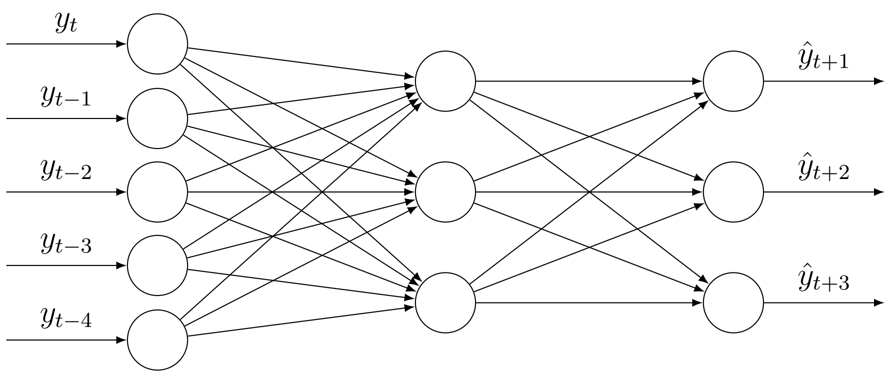
The iterative strategy has following positive features: it seeks to minimize the error of one-step forecasts, and only one model is to be fitted according to this strategy (Hyndman & Ben Taieb, 2012). However, this method is criticized to have typically less accurate forecasts for long period as it iteratively drops off the past observed values and replaces with predicted values (Zhang et al., 1998).

Direct Strategy
Following the direct strategy only one model will be fitted. In this approach the output layer consists of H neurons, as shown on Figure 9(b) for the case $H = 3$. This has advantage of using all useful past observations, with reducing the computational power needed to forecast H future values, as only one model is fitted (Zhang et al., 1998). This approach is widely used in many practical solutions in various case studies (Lee & Jeong, 2017; Montaño Moreno et al., 2011; Zhang et al., 1998).

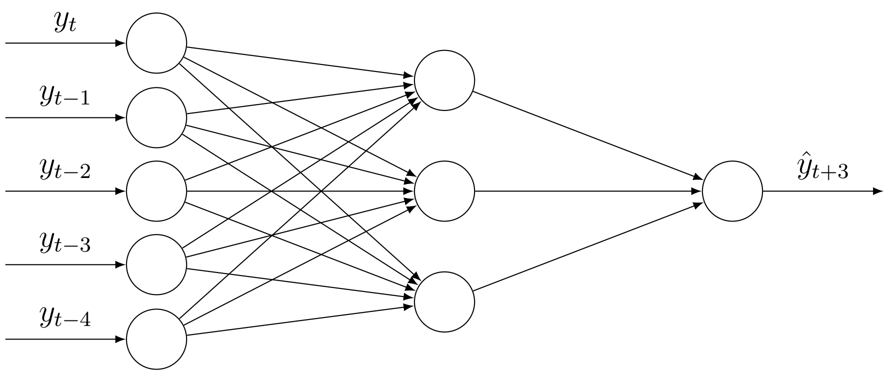
Multi-Neural Network strategy
The multi-neural network strategy, will have H separate models, one for each forecasting step h. Each model will take as input $n+1$ past observations $y_t, y_{t-1}, \dots, y_{t-n}$, and will output $\hat{y}_{t+h}$. An example of multilayer perceptron model with multi-neural network



(a) Iterative strategy



(b) Direct strategy



(c) Multi-neural network strategy

Figure 9. Examples of different strategies on multilayer perceptron architecture for forecasting the time series value $\hat{y}_{t+3}$ after $h = 3$ steps ahead given on five past observations $y_t, y_{t-1}, y_{t-2}, y_{t-3}, y_{t-4}$. On subplot (a) it is assumed that the $\hat{y}_{t+1}$ and $\hat{y}_{t+2}$ have already been forecast iteratively.

strategy to forecast value at future step $h = 3$ given on five past observations is shown on Figure 9(c).

Hyndman & Ben Taieb (2012) highlight that this strategy is computationally exhaustive, as it requires fitting multiple models. Moreover, the forecasts can be not as coherent as with iterative strategy, as the models for different horizons are estimated independently. When compared to iterative strategy, it has advantage of using all past observations for forecasting.

This strategy is sometimes also referred to as direct strategy (Ben Taieb, Bontempi, Atiya & Sorjamaa, 2012; Hyndman & Ben Taieb, 2012), but the term *multi-network strategy* will be preferred not to confuse it with previously introduced direct strategy.

Combined Strategies
It is possible combine the three strategies. For example, Zhang (1994) describes strategy that fits H models, but forecasts are made iteratively. At every iteration the input consists of all past observations $y_t, \dots, y_{t-n}$ and previously forecast values $\hat{y}_{t+1}, \dots, \hat{y}_{t+(h-1)}$, and the output of the model consists of new predictions for $\hat{y}_{t+1}, \dots, \hat{y}_{t+(h-1)}$ and a new forecast for $\hat{y}_{t+h}$.

2.4 Evaluating the Performance of a Model

When evaluating the model performance, it is possible to measure error on the same data that was used to train the model. These obtained errors are called in-sample errors. When the trained model is evaluated by the performance on yet unseen data, then the errors are referred to as out-of-sample errors.

Makridakis et al. (1982) concluded in their empirical study for time series forecasting that the model fitting errors, also referred as in-sample errors, underestimated forecasting errors on the unseen data. This conclusion has found approval by many authors according to Tashman (2000), and therefore evaluating model performance is based, by default, on out-of-sample errors.

2.4.1 Training, Validation and Test Sets

In order to have unseen data for model evaluation, the time series is split in two distinct sets: train and test set, where train data consists of sequential values until some timepoint T, and test set contains values starting from time $T + 1$ (Makridakis et al., 1982). There is no fixed rule how the data should be split, but it is common to use some proportion between training and test set, 70% vs. 30%, 80% vs. 20%, 90% vs.

10%, for example (Zhang et al., 1998). When the original series is short, then the test set should be size of input lags plus forecasting horizon.

Ord, Fildes & Kourentzes (2017: pp. 344-349) highly recommend using an extra set, commonly referred to as validation set. Validation set is obtained by dividing the training data into two subsequences, where validation set again acts as future unseen data. The combination of hyperparameters, e.g. number of hidden layers, hidden nodes, is considered as the best when corresponding model gives the lowest error on the validation set.

The authors Ord et al. (2017: pp. 344-349) bring out that validation set is used also in the neural network training phase in order to avoid overfitting. After predefined amount of iterations of the optimization algorithm, backpropagation, for example, the forecast error is calculated on the validation set. The training can be stopped when the validation error does not decrease anymore, as this indicates that further training can lead to overfitting.

Again, when separating the validation set from training data, there is no fixed rule on how to choose the proportion for the split. However, it is often similar to what was used for separating the test data.

2.4.2 Out-of-Sample Error Evaluation Methods

Tashman (2000) has compared the three following methods for out-of-sample error evaluation.

Fixed-Origin Evaluation

One approach to evaluate out-of-sample errors, is fixed-origin method. By this approach the whole input data is used only once as a whole to compute the forecast errors for horizon H. This method has a major drawback: only one error estimate can be obtained for the output, i.e., unseen data. The errors calculated in this way can be affected by the uniqueness of the unseen data, e.g. extreme fluctuations or uncommon stability, and therefore may not give the most reliable result.

Rolling-Origin Evaluation

Rolling-origin evaluation is a successive updating method, where the output data is successively added to input data after it has been assigned a one-step-ahead forecast. This way it is possible to obtain many forecasts, and the total error value obtained by many forecast errors is more reliable.

Rolling Windows Evaluation

Rolling window approach is similar to rolling-origin approach, but difference is in the length of input data. With rolling-origin method the size of input data increases after adding a value from output data. In rolling window method the size of input data is held fixed, and after adding a new output value, the oldest observation of inputs is discarded.

The neural networks do not need the previous data to be kept if the trained models are not refitted. The input size will be determined by network structure, and is equal for each time step. The previous time steps will be discarded by neural network, and therefore rolling window approach is simple yet effective approach for developing neural network models for time series forecasting.

2.4.3 Sampling Design

In practice, each of training, validation and testing subseries is transformed with rolling window method into a matrix of input values and output values (Brownlee, 2018). If a series contains values $y_1, y_2, \ldots, y_N$, and the input for model should have p past observations to forecast H future values, then the input-output matrix consists of $n = N - H - p + 1$ rows and $p + H$ columns, as shown in Table 1.

The input-output matrix obtained from training subseries is used for developing the model. The matrix obtained from validation set is used for computing the validation error, and this is the basis for choosing the best model from every modeling technique. The matrix obtained from transforming the testing set, is used for final evaluation between different architectures.

Table 1. Input-output matrix obtained from series of length N, for predicting H steps ahead based on p previous values, where $n = N - H - p + 1$.

	Input				Output			
Sample id	$y_{T-(p+1)}$	$y_{T-(p+2)}$	$\cdots$	y_T	y_{T+1}	y_{T+2}	$\cdots$	y_{T+H}
1	y_1	y_2	$\cdots$	y_p	y_{p+1}	y_{p+2}	$\cdots$	y_{p+H}
2	y_2	y_3	$\cdots$	y_{p+1}	y_{p+2}	y_{p+3}	$\cdots$	y_{p+H+1}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
n	y_n	y_{n+1}	$\cdots$	y_{N-H}	y_{N-H+1}	y_{N-H+2}	$\cdots$	y_N

2.4.4 Error Measures

There are multiple error measures used for evaluating forecasting accuracy. The following overview of the most common measures for evaluation of errors $e_t = y_t - \hat{y}_t$ is based on Hyndman & Koehler (2006).

- **Mean Absolute Error:** MAE=$\frac{1}{n}\sum_{i=1}^{n}|e_t|$, as introduced in Section 1.4.3.
- **Mean Absolute Percentage Error:** MAPE=$\frac{1}{n}\sum_{i=1}^{n}\left|\frac{e_t}{y_t}\right| \cdot 100$, as introduced in Section 1.4.3.
- **Root Mean Squared Error:** RMSE=$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_t)^2}$, the square root of MSE introduced in Section 1.4.3.

It must be taken into account that when the series $\{y_t\}$ contains values equal to 0, then MAPE measure is undefined, and when series has values close to 0 then MAPE will give extreme values.

In addition to measures given above, it is possible to combine two calculated measures to a relative measure. Often, one of the measures is calculated on some baseline model, and the other one on the alternative model. The relative measure indicates if the alternative model can be considered as an improvement over the baseline model. For example, relative RMSE is given by

$$\text{RelRMSE} = \frac{\text{RMSE}}{\text{RMSE}_b},$$

where subscript b indicates the measure calculated on baseline model. As desired error measure is as low as possible, the relative measure can be interpreted in the following way. If the relative measure is less then one, the alternative is performing better then baseline model, and if the relative measure is greater than 1, the alternative model does not give better forecast.

When choosing a best performing model amongst multiple neural network models that give error estimates of the same magnitude, Goodfellow et al. (2016: 112) advise to invoke a principle referred to as Occam's razor. Based on this principle a simpler model should be chosen, if the models describe the data equally well.

2.5 Approaches for Improving the Neural Network Model Performance

The following section is based on Goodfellow et al. (2016: pp. 117, 226-231, 253-262).

2.5.1 Regularization

The generalization error, that is the error which a model makes to predict on the new unseen data, increases with overfitting. In addition to using validation set for stopping the training process when the generalization error starts to increase, there are more possibilities for keeping overfitting under control.

Regularization is any kind of modification in the learning algorithm with the aim to lower the generalization error but not the training error.

Parameter Regularization

The following parameter norm penalty ideas, L^1 and L^2 regularization, are used widely in linear models. In neural network learning algorithm the penalized parameters are only the weights $\mathbf{W}$ connecting neurons, and not the bias parameters $\mathbf{b}$. Recall, $\boldsymbol{\theta}$ denotes both, now penalized weights and regular, unpenalized biases.

The penalty effect is obtained by adding a regularization term $\Omega(\boldsymbol{\theta})$ to the cost function $\mathcal{E}(\boldsymbol{\theta})$ that is to be minimized. Therefore, the regularized cost function $\tilde{\mathcal{E}}(\boldsymbol{\theta})$ is given by

$$\tilde{\mathcal{E}}(\boldsymbol{\theta}) = \mathcal{E}(\boldsymbol{\theta}) + \alpha\Omega(\boldsymbol{\theta}),$$

where α is a hyperparameter regulating the contribution of regularization term.

In case of L^2 regularization, the added regularization term $\Omega(\boldsymbol{\theta})$ is following:

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\left\|\mathbf{W}\right\|_2^2 = \frac{1}{2}\sum_{w\in\mathbf{W}} w^2,$$

and in case of L^1 regularization, the added regularization term $\Omega(\boldsymbol{\theta})$ is following (Nielsen, 2015):

$$\Omega(\boldsymbol{\theta}) = \left\|\mathbf{W}\right\|_1 = \sum_{w\in\mathbf{W}} |w|.$$

Dropout

Another regularization method, dropout, is a method of dropping non-output neurons from the network. In case of learning algorithm with minibatches, like SGD introduced in Section 1.4.2, the neurons chosen to be excluded are drawn randomly with chosen probability for each of the minibatches. The probability of including a neuron is again a hyperparameter that can be tuned manually to find the best performing model. It is shown that this method reduces the generalization error, especially of deep neural networks, that often suffer from overfitting. However, to obtain the level of lower validation error has a cost of training a larger network and large number of iterations in the learning algorithm.

2.5.2 Ensemble Models

It is possible to reduce error by combining output of multiple separate models. The motivation for the approach is that usually, different models will make different errors on the test set.

Ensemble models can be constructed in many ways. It can contain the models constructed with exactly the same architecture. In this case different output is obtained by introducing randomness in the initialization of parameters, and/or in the choice of minibatches, for example. In addition, an ensemble model can be constructed of models with different architecture.

Kourentzes, Barrow & Crone (2014) advise using median or mode operators for combining results from multiple models. They emphasize that mean operator is sensitive to outliers and has poor performance when the distribution of outputs is skewed. The authors demonstrated in their empirical study the superiority of median operator when compared to mean operator. Furthermore, they proposed mode operator for combining the models. As a result, the authors concluded that mode operator results in lower error compared to median operator when the ensemble contained of at least 30 individual models. However, as the improvement over median operator was not remarkable, median operator is still more popular.

2.5.3 Hybrid Models

Several empirical studies have introduced different hybrid models. In a hybrid model, the time series is often decomposed as linear and nonlinear component, i.e.,

$$y_t = N_t + L_t,$$

where N_t is nonlinear component and L_t linear component.

Zhang (2002) proposed a hybrid model that, first, fits a linear ARIMA model to the series, and second, fits a neural network model to the nonlinear residuals. Adhikari & Agrawal (2013) proposed a similar hybrid model, where the linear part was fit with naïve model. Both studies affirmed the superiority of hybrid model over both linear and nonlinear model separately.

Several other hybrid models have been proposed. For example, a hybrid of LSTM and convolutional neural network, that were mentioned in Section 1.3. (Lin, Guo & Aberer, 2017).

3 Empirical Study for Comparing the Time Series Forecasting Models

An empirical study was conducted to compare neural network forecasting performance with classical linear models on different datasets.

The empirical study for comparing time series forecasting models will follow the given structure. Section 3.1 gives an overview of the packages of programming languages `Python` and `R` used for training the neural networks, Section 3.2 describes in detail how the models are trained, and how it is determined which model has the best performance, Section 3.3 introduces briefly the datasets used in comparison, Section 3.4 presents the chosen models and obtained results for each dataset, and finally, Section 3.5 analyzes the results across datasets, packages and model architectures.

3.1 Overview of Suitable Packages for Forecasting in `R` and `Python`

The following section covers different possibilities in forecasting with neural networks, primarily with statistical programming language `R`. In addition, programming language `Python` is used for developing LSTM models. It must be noted, that the following list of packages is incomplete, and will cover only a handful of possible options. The used packages were chosen because they offered flexibility for customizing the model structure, and allowed the neural networks to be compared with rolling window approach introduced in Section 2.4.2.

The packages introduced below differ by the speed of training process, supported modeling techniques, and the implemented architectures.

3.1.1 Package `neuralnet` for `R`

Package `neuralnet` is general package for training feedforward neural networks in `R`. The first version of the package was developed in 2008 by Fritsch, Guenther & Wright (2019). This package does not have any special functions for forecasting time series, so the task can be implemented by multilayer perceptron architecture using different modeling strategies introduced in Section 2.3.2.

However, the duration of training process can limit the choice of possible network architectures. Firstly, the computations in `R` are by default single-threaded, and the package itself has not implemented multi-threaded calculations. Secondly, the function for training the network does not offer using faster, currently more popular and more advanced optimization algorithms, for example, stochastic gradient descent introduced in Section 1.4.2 (Fritsch et al., 2019, Goodfellow et al., 2016: pp. 307).

3.1.2 Package `h2o` for `R`

Package `h2o` is an interface for program 'H20' in `R`, offering diverse selection of implementations for different machine learning algorithms (LeDell et al., 2019). From neural network modeling it supports training deep feedforward networks. Similarly to package `neuralnet`, this package does not have any special functions for forecasting time series, so the forecasting networks can be trained according to iterative and multi-neural network strategy introduced in Section 2.3.2. Currently, package `h2o` does not offer possibility to model according to the direct strategy shown in Section 2.3.2.

The function `deeplearning`, used for fitting deep neural networks, offers wide selection of hyperparameters that can be tuned to achieve more precise model. Moreover, the computations made by functions of package `h2o` can be run on multiple threads on CPU, or even GPU, and this accelerates the training process remarkably.

The package offers also an automated grid search function for improving the procedure of hyperparameter tuning.

3.1.3 `Keras` for `Python`

`Keras` is a deep learning library written in `Python` by F. Chollet in 2015 that allows developing very flexible and advanced deep learning models (Chollet, 2015). Additionally, it has an interface to `R`, and can be run by using package `keras` (Allaire & Chollet, 2019). However, the code for developing models in this thesis were written in `Python`.

Deep learning has been made very flexible by using `Keras`, as it offers constructing networks layer by layer, while specifying the details separately for each layer. It offers variety of layer types, and the possibility to build a custom layer. The library supports recurrent neural networks, and therefore it was used to develop long short-term memory models for time series forecasting.

`Keras` can be run on both CPU and GPU, enabling to model large-scale tasks.

3.2 Design of the Comparative Empirical Study

For comparison, there were multiple models fitted for each dataset that will be introduced in Section 3.3. The empirical study was executed for each dataset in following three steps: fitting baseline models, choosing neural network architectures with the best performance, and lastly, choosing the neural networks with the best performance for each dataset and forecasting horizon H.

3.2.1 Generating input and output datasets

First, the time series is split into three subseries for training, validation and testing processes. The rule for all data sets was following: 15% of the series (most recent observations) were set apart for testing the performance of chosen models. The remaining series was split into two – the first 80% of the values were used as training set, and the last 20% of the observations were used as validation set.

Afterwards, each of the three obtained subseries was transformed into input and output matrices by using rolling window approach as introduced in Section 2.4.2 and Section 2.4.3.

3.2.2 Linear Models

To demonstrate the success or failure of forecasting with the developed neural networks, they must be compared against some baseline model. Both naïve and ARIMA, or seasonal naïve and SARIMA models were fitted for comparison of the neural network performance.

The ARIMA models used as baseline models were fitted with `R` function `auto.arima`. Therefore, these models can be considered as good candidates for best performing ARIMA model. The emphasis of this thesis was to discover how to use neural networks in time series forecasting, and to see whether applying these approaches can offer results with some practical value. Thus, as the aim was not to find the overall best ARIMA models for chosen datasets, the ARIMA models fitted with `auto.arima` were considered good enough for baseline models, and no benchmark of ARIMA models was attempted to find. However, for simplicity, the models that were chosen by `auto.arima` are referred to in this thesis as "the best" models, hereby fully acknowledging that this may not hold in all cases.

The orders for ARIMA model were chosen on training set, and moreover the parameters were estimated on training set as well. This means that the models were not refitted before each forecast. By following this approach, the settings for both linear and neural network models were fair, in the sense of equal prior knowledge.

3.2.3 Neural Network Models

After obtaining linear baseline models, a manual grid search was performed for finding the best neural network architecture for given input and forecast horizon. Grid search iterated over different combinations of forecasting horizons, number of input variables, modeling strategies.

As a result of training process, each model had obtained parameter estimates based on training data. As refitting the more complex neural network models is costly, the classical approach of machine learning was followed, and thus the model parameters were not refitted after obtaining new information from validation and test set.

Three packages introduced in Section 3.1 were used for developing neural network models. The specifications for each package is given in paragraphs below.

The obtained multilayer perceptron networks were partitioned into two groups, small networks and large networks by the number of hidden neurons. Small MLP networks had less than 20 hidden neurons in total over all hidden layers. This partition gave better overview for deciding whether the forecasting performance can be improved by using more complex models.

All compared models were ensembles of 10 networks with identical structure, producing a forecast $\hat{y}_{T+h}$ as the median value of individual forecasts, where $h = 1, \ldots, H$. The ensemble size was chosen based on Kourentzes, et al. (2014), where the empirical analysis showed that by using ensemble with median forecast obtained the low error already by small ensemble. Of course, the exact number depends on the dataset used, but to limit the time spent on training, it was not reasonable to use ensemble sizes greater than 10 for all datasets and all functions.

Forecasting with Package `neuralnet`

As package `neuralnet` does not explicitly allow multi-threaded computations, and therefore takes long time to train, only small networks were developed using this package. The time series modeling strategy for `neuralnet` models was direct approach.

The models were trained with the resilient backpropagation algorithm - the default of this package, and it minimized the L2 loss function (Fritsch et al., 2019). The activation function for hidden layers was by default sigmoid function. A million steps were allowed for the training algorithm to converge into some local minima, and if the threshold was surpassed, the training process was stopped without obtaining a model.

Forecasting with Package `h2o`

With package `h2o` there were two types of models fitted, first were based on multi-neural network strategy, and second followed the iterative strategy.

The `h2o` uses by default the stochastic gradient decent algorithm with additional improvements to accelerate the convergence of the algorithm (Candel & LeDell, 2019), which minimized the L2 loss function. Most of the hyperparameters for the training function `deeplearning` were used with the default values, among them also rectified linear unit activation function. However, to improve model training process and ability to generalize, two of the hyperparameters were manually tuned. First, early stopping based on validation set error measure was enabled. Second, L^1 regularization was used with rate $\alpha = 0.0001$ (as defined in Section 2.5.1), as using this resulted in lower error measure, when testing it with multiple datasets.

Forecasting with `Keras`

Long short-term memory models were developed with package `Keras`. These models were trained with direct strategy, i.e., the output layer contained H neurons. The used optimization algorithm is called Adam which is a further development of stochastic gradient decent algorithm, where the learning rate is changed adaptively. Like with `h2o` models, the loss function was MSE, and early stopping was enabled. Additionally, input layer dropout was tried, but no good rate of dropout were detected manually, and therefore not used.

Developed long short-term memory models had 10, 20 and 100 nodes for each H and p combination.

3.2.4 Choosing the best neural network models

When using the best linear models time series forecasting, the best models are often chosen based on residual diagnostics, and Ljung-Box test results (Hyndman & Athanasopoulos, 2018). However, it is possible to evaluate the goodness of a model based and testing the model on some new data. In this thesis the latter approach is used.

For each linear and neural network model the error measure was computed as follows.

Let N denote the length of time series. Let $\hat{\mathbf{Y}}$ denote obtained matrix of forecasts, and $\hat{\mathbf{y}}_h$ denote the h-th column vector of $\hat{\mathbf{Y}}$.

As shown in Section 2.4.3, the time series with length of N can be transformed into input and output matrices with $n = N - H - p + 1$ rows. The length of input and output matrix of test subseries is therefore $n_t = \lfloor 0.15N \rfloor - H - p + 1$, and length of obtained matrix for validation subseries is $n_v = \lfloor 0.2(\lceil 0.85N \rceil) \rfloor - H - p + 1$.

The obtained forecasts for validation and test set were following:

$$\hat{\mathbf{Y}}^{val} = \begin{bmatrix} \hat{y}_{11}^{val} & \hat{y}_{12}^{val} & \hat{y}_{13}^{val} & \cdots & \hat{y}_{1H}^{val} \\ \hat{y}_{21}^{val} & \hat{y}_{22}^{val} & \hat{y}_{23}^{val} & \cdots & \hat{y}_{2H}^{val} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{y}_{n_v1}^{val} & \hat{y}_{n_v2}^{val} & \hat{y}_{n_v3}^{val} & \cdots & \hat{y}_{n_vH}^{val} \end{bmatrix},$$

$$\hat{\mathbf{Y}}^{test} = \begin{bmatrix} \hat{y}_{11}^{test} & \hat{y}_{12}^{test} & \hat{y}_{13}^{test} & \cdots & \hat{y}_{1H}^{test} \\ \hat{y}_{21}^{test} & \hat{y}_{22}^{test} & \hat{y}_{23}^{test} & \cdots & \hat{y}_{2H}^{test} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{y}_{n_t1}^{test} & \hat{y}_{n_t2}^{test} & \hat{y}_{n_t3}^{test} & \cdots & \hat{y}_{n_tH}^{test} \end{bmatrix}.$$

As the next step, error measure RMSE was calculated for each h-th column, $h = 1, \ldots, H$, to measure the error of h-steps ahead forecasts, i.e, RMSE($\hat{\mathbf{y}}_h$).

If the forecasting horizon H was longer than one step, then the comparable error measure was obtained by averaging error measures over all horizon steps h. For example, average RMSE for test set forecasts was calculated as

$$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{test}) = \frac{1}{H} \sum_{h=1}^{H} \text{RMSE}(\hat{\mathbf{y}}_h^{test}).$$

The neural networks were compared with baseline models based on the error measures calculated from validation set.

It must be noted that both linear and nonlinear models that were compared by absolute number, had to be evaluated on the same time interval. This avoids the possible unfair effect by extreme values or irregular stability, for example. Therefore, the relative measures, introduced in Section 2.4.4, could only be calculated for each H and p combinations separately. This approach gave more reliable results when the series contained extreme changes in the behavior, which will be discussed in more detail in Section 3.4.5.

Based on the relative measures on the validation set, the best neural networks were chosen for each combination of forecasting horizon, used package and modeling strategy. The best neural network demonstrated the highest improvement over linear methods. The final evaluation of chosen models was done by comparing the results on test set.

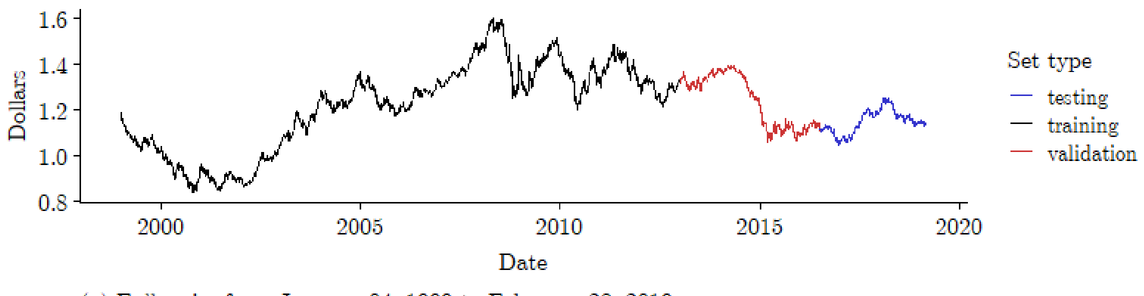
3.3 Datasets

All chosen datasets vary in the frequency of the observations, appearance of seasonal component and nonlinear components.

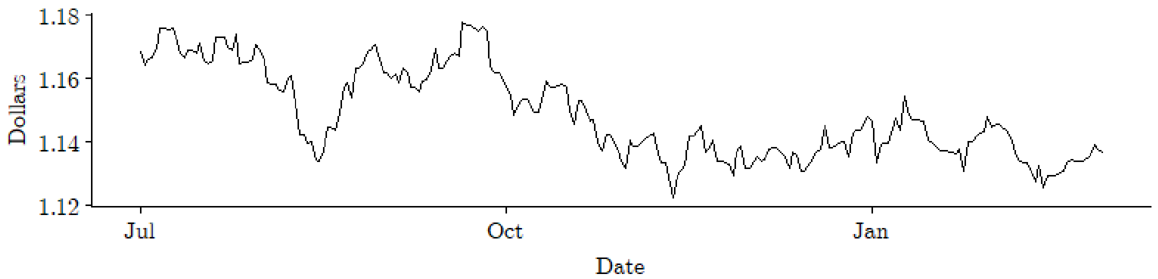
The datasets were divided into training, validation and test sets, and the splits for each dataset are depicted on figures below. All the datasets were standardized before training and using trained models for forecasting. When the data contained trend component, the series was differenced. As the neural networks try to fit the observable patterns, no seasonal diffencing was performed that would to remove the seasonal patterns.

3.3.1 Euro - US Dollar Exchange Rate

The first analyzed dataset contained EUR-USD exchange rates. The exchange rates are collected by Macrotrends LLC, and were retrieved for analysis from their website (Macrotrends LLC, 2019). They have published daily closing prices starting from January 4, 1999. The exchange rates were given on market days, so weekends and holidays were mostly not present in the dataset. Until 2018 both Saturdays, and Sundays were excluded, afterwards Saturdays were included, and additionally Sundays were observed from February 25, 2018 to September 9, 2018. The analyzed data contained 5194 values from January 4, 1999 to February 28, 2019 (see Figure 10).



(a) Full series from January 04, 1999 to February 28, 2019.



(b) Subseries from July 01, 2018 to February 28, 2019.

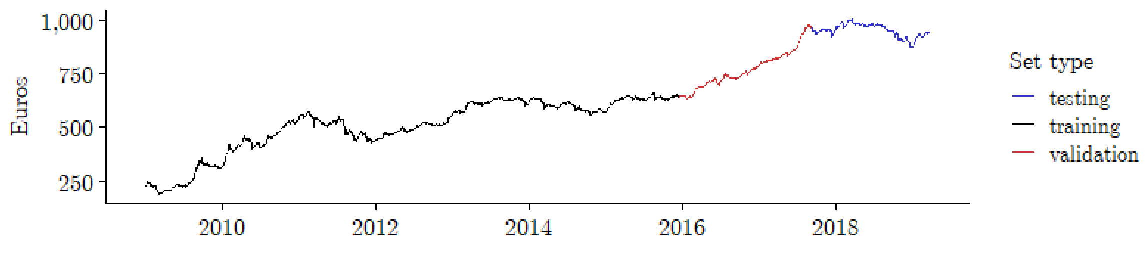
Figure 10. Daily exchange rate of 1 EUR in US dollars.

From the Figure 10(a) it can be seen that the data has no long term trend, however it contains multiple local trend direction changes. Additionally, it can be seen from Figure 10(b) that the series does not have a regular seasonality component. The series was not differenced for developing neural network models.

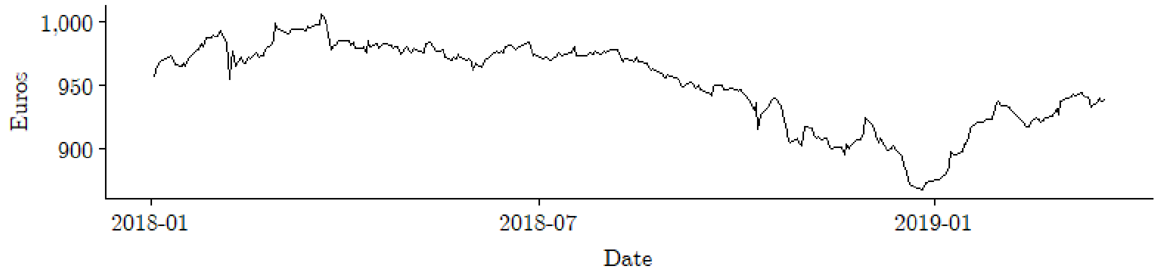
3.3.2 OMX Baltic Benchmark Index

The second analyzed dataset contained the OMX Baltic Benchmark index, that is calculated over large stocks with high trading activity on Nasdaq Baltic Market. The index is owned and calculated by Nasdaq, and is published on Nasdaq Baltic website (Nasdaq Baltic, n.d.-a). The index is calculated in euros, with the base value of 100€ on December 31, 1999. The methodology of calculations has been published by Nasdaq Baltic (Nasdaq Baltic, 2016). The index prices are given only on market days, i.e, from Monday to Friday.

The data was retrieved from Nasdaq Baltic website (Nasdaq Baltic, n.d.-b) and contained observations from January 1, 2009 to March 22, 2019, in total of 2587 observations (see Figure 11).



(a) Full series from January 01, 2009 to March 22, 2019.



(b) Subseries from January 02, 2018 to March 22, 2019.

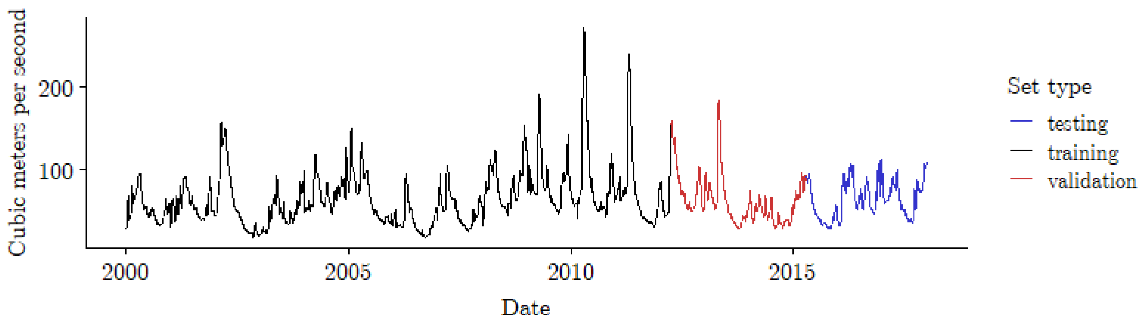
Figure 11. Daily price of OMX Baltic Benchmark index in euros.

It can be seen from Figure 11(a) that the data has long term trend, and additionally, it can be seen from Figure 11(b) that the series does not contain a seasonality component. The series was differenced for developing neural network models.

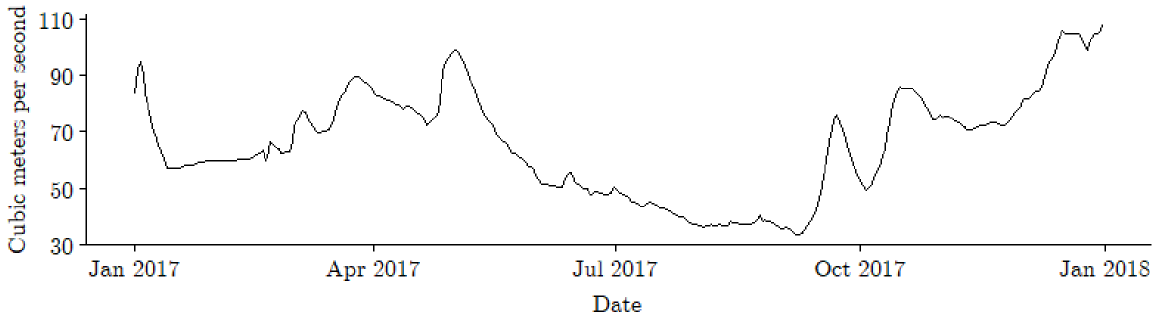
3.3.3 River Flow of the Great Emajõgi

The third analyzed dataset describes the river flow in m^3/s of the river Great Emajõgi measured in Tartu, South Estonia. The data is collected by Estonian Weather Service, and was retrieved from their website (Estonian Weather Service, 2019). The data has been collected daily starting from the beginning of January 1922. The data used in analysis contained 6575 values from January 1, 2000 to December 31, 2017 (see Figure 12).

From the Figure 12 it can be seen that the data has irregular behavior, no trend and no weekly or monthly seasonality can be observed. However, some years have high peaks caused by floods at spring time. As the series did not have trend, the series was left as original, no differencing was done for modeling purposes.



(a) Full series from January 01, 2000 to December 31, 2017.



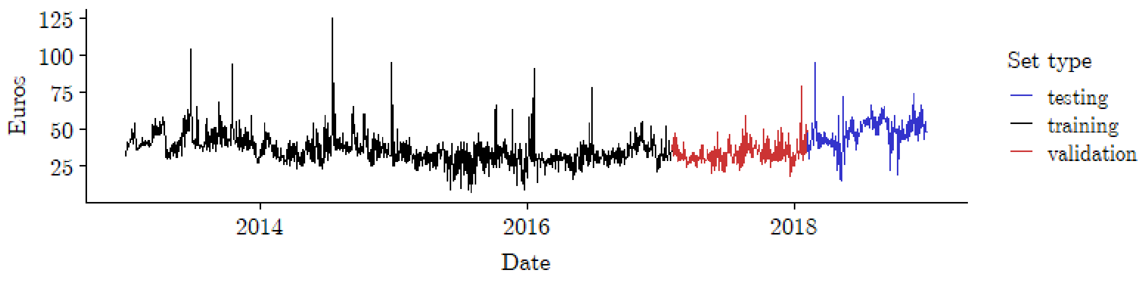
(b) Subseries from January 01, 2017 to December 31, 2017.

Figure 12. Daily river flow of river the Great Emajõgi, measured in m^3/s.

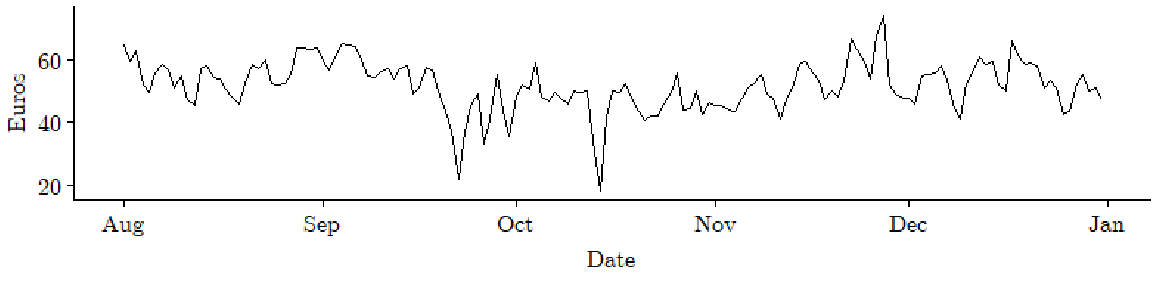
3.3.4 Electricity prices in Estonian power market

The next analyzed dataset contained electricity prices at Nord Pool power market for Estonia. The data is collected by Nord Pool AS, and was retrieved for analysis from their website (Nord Pool AS, 2019). They have published historical data in various frequencies starting from the beginning of January 2013.

The analyzed prices are based on the Elspot day-ahead trades in Nord Pool power market for region Estonia. Only daily prices were chosen for further analysis. The used data contained 2191 values from January 1, 2013 to December 31, 2018 (see Figure 13).



(a) Full series from January 01, 2013 to December 31, 2018.



(b) Subseries from August 01, 2018 to December 31, 2018.

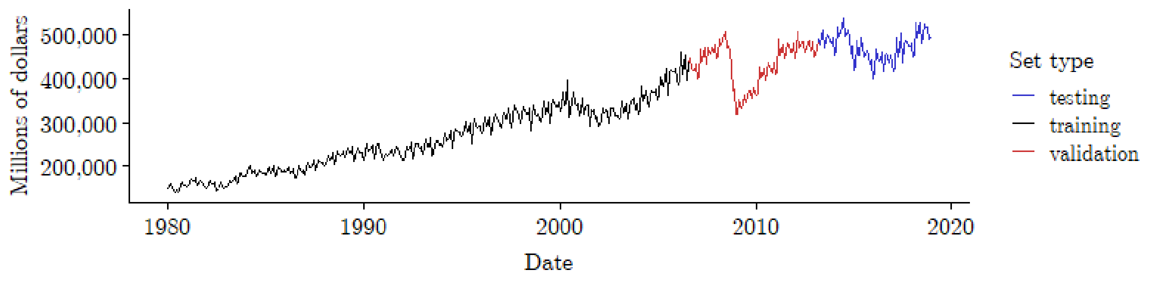
Figure 13. Daily electricity prices in Estonia, measured in euros.

From the Figure 13(a) it can be seen that the data has no trend, and from Figure 13(b) that there is weekly seasonality in some extent. This is confirmed by autocorrelations and partial autocorrelations given in Appendix C. The series was differenced for developing neural network models, but the calculated errors were based on the original data.

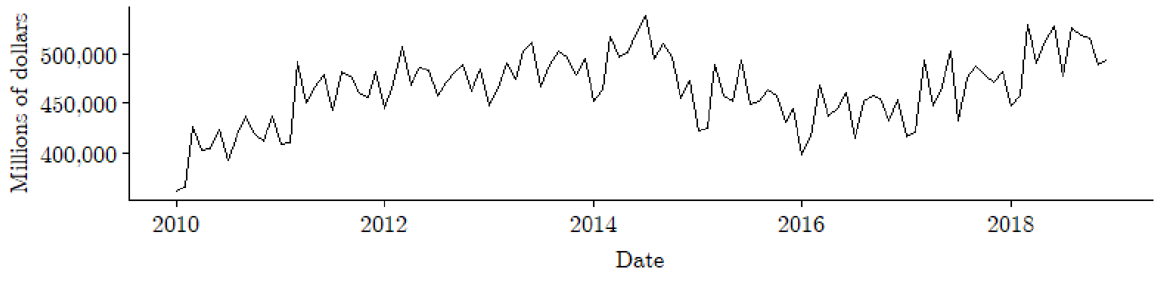
3.3.5 New Orders in the Manufacturing Sector in the U.S.

The last analyzed dataset described the sum of new orders (in millions of dollars) received, net of cancellations, in the manufacturing sector in the United States. The establishments considered to be in manufacturing sector transform materials, substances or components into new products.

The data is collected from domestic manufacturing companies by the U.S. Census Bureau, and data for analysis was retrieved from their website (U.S. Census Bureau, 2018). The data is collected monthly from February 1958. The data used in analysis contained 468 values from January 1980 to December 2018 (see Figure 14).



(a) Full series from January, 1980 to December, 2018.



(b) Subseries from January, 2010 to December, 2018.

Figure 14. Monthly total of new orders in manufacturing sector in the U.S., measured in millions of dollars.

From the Figure 14 it can be seen that the data has increasing trend. Figure 14 and figure in Appendix C indicate seasonal component, thus, the series was differenced for modeling purposes. However, the calculated errors were based on the original, undifferenced data.

As the chosen series describes economic data, it has been greatly affected by recessions. The 2008 financial crisis had strong impact on the manufacturing sector, and as one of the consequences was a sudden drop in new orders in this sector (Figure 14). Although the impact was strong, and affected forecasting models in a great extent, it was not taken into consideration to change the data in some way, e.g., by leaving out the financial crisis period, or scaling down the effect. In this way it can be an example of how the neural networks are affected by these extreme changes with high impact on upcoming time points.

3.4 Results

The best models obtained by every method (different package, small vs large network, modeling strategy) are given in appendices, the overall best neural network models are highlighted within sections.

Common remarks and conclusions for all datasets are given as follows.

- The error measures calculated for each h-step ahead, $\text{RMSE}(\hat{\mathbf{y}}_h)$, generally increased with h.
- For most of datasets only one hidden layer was enough, additional layers decreased performance.
- LSTM models had very long training time. However, this is affected by overall low CPU utilization by `Keras`.
- Despite the fact that the networks trained with package `neuralnet` were not very deep, the training time was very long. First, it was in great extent due to single-threaded calculations. Furthermore, the optimization algorithm had problems with converging within million iteration steps. For many models the algorithm did not converge at all, and corresponding model was not achieved.
- As the training time of models with iterative method increased faster than the multi-neural network models, and did not obtain the same level of improvement over baseline, the iterative models were not trained in case of long horizon H or large number of inputs p.
- One-step-ahead forecasts obtained with `h2o` by iterative and multi-neural network strategy, are calculated in the same manner, and therefore had similar results. When the difference between strategies is compared, the conclusions are made on basis of forecasts for longer horizon H, as these statements do not hold in case of $H = 1$.
- When comparing `h2o` models by iterative and multi-neural network strategies, the results indicated the superiority of multi-neural network method.
    1. Although the results on validation set were in same size for both strategies, the results on test set indicated strongly that the iterative method overfitted to validation set. Therefore, this strategy will perform worse on unseen future data when compared to multi-neural network method.
    2. The elapsed time for training the model increases remarkably faster for modeling more and more complex networks with the iterative strategy.

The best models based on validation set are highlighted in comparison tables below for each analyzed dataset. All the tables have similar structure: they demonstrate how long input was chosen and how many neurons were used by the best models for specific package, modeling strategy and forecasting horizon H. For each best model the error measure on validation set $\overline{\text{RMSE}}\big(\hat{\mathbf{Y}}^{val}\big)$ or $\overline{\text{MAE}}\big(\hat{\mathbf{Y}}^{val}\big)$ and corresponding value on test set $\overline{\text{RMSE}}\big(\hat{\mathbf{Y}}^{test}\big)$ or $\overline{\text{MAE}}\big(\hat{\mathbf{Y}}^{test}\big)$ are given. Percentages show the relative error when compared with baseline model. Negative percent demonstrates by how many percentages lower error was obtained by the neural network. Positive percent shows how much larger the error was when compared to baseline model. Training time is given in seconds. Additionally, a rank within the group of similar models was given based on the result on the test set. All the models highlighted in tables have the lowest validation error, thus always the rank of 1 on validation set, and hence the validation rank was discarded.

The test rank gives information about the consistency of chosen models. An ideal model would rank as first on both validation and test set. In this way, the comparison results are more reliable, as the model could be considered as a model that is consistently better when forecasting new unseen data. However, as the hyperparameters tuned did not significantly change the behavior of neural networks, the differences in error measure were mostly not remarkable, and highly ranked models could all be considered as good models, not just the model with rank 1.

It must be noted that the given training times are only rough estimates. The conditions at measuring were not fair and equal for all models, as many models were training at the same time by `R` and `Python`.

3.4.1 Forecasting Euro - US Dollar Exchange Rate

As EUR-USD exchange rate dataset had daily frequency with no seasonal component, the forecasting horizon H was chosen to be equal to 1, 3, 5, 10, 15. The values for p, number of previous observations, were calculated as multiple of H with factors 2, 5 and 10.

First, linear baseline models were fitted. The best ARIMA model for all H and p combinations was ARIMA(0,1,0), and therefore, ARIMA model is equal to naïve model. Afterwards, manual grid search was performed to find neural network models that could offer an improvement over naïve model.

The best results among all neural network models, regardless of the package used, were obtained by networks that used $p = 2 \cdot H$ inputs.

The best performing neural network models for all forecasting horizons for EUR-USD exchange rate are given below in Table 2. A comprehensive overview with the best models for every package is given in Appendix D.

Table 2. Comparison of the best performing neural network models with naïve forecasts for EUR-USD exchange rate. All models had the lowest error on both validation and test set.

Model			Hidden	Train	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{val})$	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{test})$	
type	H	p	neurons	time (s)	Neural network	Naïve	Neural network
LSTM	1	2	20	604	*0.00548 (-17.90%)*	0.00485	**0.00417** ***(-14.00%)***
	3	6	20	603	*0.00842 (-6.88%)*	0.00652	**0.00640** ***(-1.74%)***
	5	10	10	878	*0.01025 (-5.64%)*	0.00783	**0.00776** ***(-0.92%)***
	10	50	10	7512	*0.01361 (-6.81%)*	0.01040	**0.01010** ***(-2.92%)***
	15	30	10	1623	*0.01539 (-11.41%)*	0.01238	**0.01194** ***(-3.57%)***

Multilayer Perceptrons with Package `neuralnet`

The models developed on package `neuralnet` were chosen to be one-layer networks, and had $3, 4, \ldots, 9, 10$ neurons in the hidden layer. Additional layers demonstrated poor performance, and were excluded from further analysis.

Not many models were obtained, none of the could remarkably improve over the naïve model. The developed models were mostly for one-step-ahead forecasts. The networks for longer horizon H did not converge in limited iteration steps or reasonable time.

Neither the number of hidden neurons nor the number of preceding observations used as input had distinctive affect. Therefore, it would be enough to use very simple architectures, e.g. with 3 or 4 hidden neurons.

Multilayer Perceptrons with Package `h2o`

For every horizon H and number of input values p, the chosen hidden layers and hidden neurons were the following: one-layer networks with $3, 4, \ldots, 9, 10, 20, 50, 100$ neurons, and a two-layer network with 10 neurons on both layers.

Both iterative and multi-neural network strategies were tried. But as mentioned in the overview of common conclusions at the beginning of Section 3.4, the training time of iterative models increased rapidly, and moreover, these models had very poor performance on test data (see Appendix D).

Small multilayer perceptron networks demonstrated lower error when compared to large networks. This may indicate that the data did not contain recurring patterns to a greater extent, and less neurons were needed to describe these. Nevertheless, even the best small networks could not improve over baseline model. The longer the horizon H, the higher error rate when compared with naïve model, as it can be seen from Appendix D. The results of small networks trained with multi-neural network strategy were on the same level with the direct strategy models obtained with package `neuralnet`.

Long Short-Term Memory Models with Package `Keras`

LSTM models performed the best with 10 or 20 cells, the larger network with 100 nodes mostly did not perform as well as smaller networks, and none of the times remarkably better than smaller models. When the forecasting horizon was short, i.e., 1 or 3 days, then models with 20 nodes obtained better results, on the contrary when the forecasting task was more complex with longer horizon H, the smallest networks with 10 neurons had better performance.

The LSTM models produced the overall lowest errors both on test and validation set, however, with the cost of high computing time. For example, to train a `h2o` MLP with 10 neurons it took 19 seconds, and to train a MLP model with 100 neurons it took 21 seconds, whereas LSTM models trained roughly 9 and 21 minutes correspondingly, i.e. about 26 and 60 times longer.

As LSTM models were the only models that showed improvement over linear methods, these models were chosen to be the best neural network models for EUR-USD exchange rate forecasting.

3.4.2 Forecasting OMX Baltic Benchmark Index

Similarly to EUR-USD exchange rate dataset, OMX Baltic Benchmark index data has daily frequency with no seasonal component. Therefore, the forecasting horizon H was chosen to be equal to 1, 3, 5, 10, 15. As there is no visible seasonal component, the values for p were calculated as multiple of H with factors 2, 5 and 10.

First, ARIMA and naive models were fitted. Obtained model by function `auto.arima` was ARIMA(1,1,0). This model had better performance on validation set than naïve forecast, and therefore chosen as baseline model.

Afterwards, manual grid search was performed to find neural network models that could offer an improvement over ARIMA model. The networks were trained on differenced series. Overall, the performance of the most trained neural networks was worse than the performance of the ARIMA method. As the differenced series in financial domain follow random walk (Hyndman & Athanasopoulos, 2018), this can be the reason for poor performance of multilayer perceptrons when there are no patters to detect.

The training process of models with different H and p combinations did not converge when modeled with `neuralnet`. Therefore, this package will not be covered in following analysis. Additionally, more complex LSTM models had the same issue, and no models were obtained for those.

Interestingly, more complex models performed better on validation set, but the models were not consistent. The models that had great performance on validation set, had much worse error measure on test set. In addition, in all cases, forecasting short horizon, e.g. 1, 3 steps ahead, produced smaller loss (or even in some cases higher improvement) when compared to baseline model.

Despite the fact that some LSTM models did not converge, other LSTM models had the best and most consistent models overall. These models are highlighted below in Table 3, a comprehensive overview of the best models for every package on OMX Baltic Benchmark index data is given in Appendix E.

Table 3. Comparison of the best performing neural network models with ARIMA forecasts for OMX Baltic Benchmark index.

Model			Hidden	Train	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{val})$	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{test})$	
type	H	p	neurons	time (s)	Neural network	ARIMA	Neural network
LSTM	1	10	10	1189	*3.188 (-2.11%)*	4.494	**4.414 (-1.76%)**
	3	15	20	1417	*4.405 (-5.17%)*	6.233	**6.147 (-1.37%)**
	5	25	10	1977	*5.426 (-7.65%)*	7.738	**7.728 (-0.12%)**
	10	20	10	643	*7.739 (-11.27%)*	**10.640**	10.779 (1.30%)
	15	30	20	1050	*9.990 (-13.16%)*	**13.001**	13.191 (1.46%)

Multilayer Perceptrons with Package `h2o`

For every horizon H and number of input values p, the chosen hidden layers and hidden neurons were following: one-layer networks with $3, 4, \dots, 9, 10, 15, 50, 100$ neurons.

The performance of iterative strategy models was similar to all other datasets - the results on validation set were promising, but unfortunately the models were not stable enough to reach the same level on new unseen data. In case of multi-neural network strategy models, both small and large networks had similar error measures, but none of these could improve over baseline on the test set.

Long Short-Term Memory Models with Package `Keras`

LSTM models performed the best with 10 or 20 cells, but in most cases also models with 100 nodes were on same level, with some exceptions.

When the forecasting horizon was short, i.e., 1, 3 or 5 days, most of the models offered improvement over baseline. Furthermore, these were the only models to reach this level. Therefore, the LSTM models are highlighted for having the best performance on both validation and test set for forecasting OMX Baltic Benchmark index (Table 3).

3.4.3 Forecasting River Flow of the Great Emajõgi

The chosen horizons H for forecasting the river flow of the Great Emajõgi were 1, 3, 7, 14 days ahead. The input p was chosen to be 7, 14, 28, 35, 42.

The dataset has irregular behavior and great variance in the values. Thus, in order not to punish the models too much for making large errors, the error measure for this dataset was chosen to be mean absolute value instead of sum of squares metric. This was used as a cost function (Section 1.4.3) in the neural network training process, and likewise as error measure (Section 2.4.4, Section 3.2.4) for comparing developed models.

First, ARIMA and naïve models were forecast. As expected, the naïve method did not suit for this dataset that demonstrated high fluctuations. The best ARIMA model obtained was ARIMA(1,1,2), and therefore, it was the model chosen as the baseline model for neural networks.

From Figure 12 it can be seen that validation set has higher peaks indicating intense river flow when compared to test set. Therefore, it would be natural to assume that the error measure on validation set is higher. Instead, most of the models produced lower error on validation set. With neural networks it is possibly due to using validation set as feedback for terminating the training process to avoid overfitting. However, for ARIMA model it still remains intriguing, furthermore as the error rates are lower when using RMSE for comparison.

Unsuccessfully, the training process of `neuralnet` models terminated without converging to a minima, and no models were obtained.

Overall, neural networks had difficulties in improving over the ARIMA model. Most of the errors by neural networks were on the same level or much higher than the results by baseline model. The only models that produced lower errors were LSTM models, and the results were consistent on both validation and test set, and furthermore did not depend as much of the parameters chosen.

The LSTM models with the lowest error measures for all horizons H are given below in Table 4, a comprehensive overview with the best models for every package and strategy is given in Appendix F.

Multilayer Perceptrons with Package `h2o`

For every horizon H and number of input values p, the chosen hidden layers and hidden neurons were following: one-layer networks with $2, 3, \dots, 9, 10, 100, 200$ neurons, and a

Table 4. Comparison of the best performing neural network models with ARIMA forecasts of river flow of the Great Emajõgi.

Model type	H	p	Hidden neurons	Train time (s)	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{val})$ Neural network	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{test})$ ARIMA	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{test})$ Neural network
LSTM	1	35	100	1077	*0.750 (-1.04%)*	0.881	**0.880 (-0.08%)**
	3	35	100	907	*1.588 (-3.85%)*	1.889	**1.869 (-1.07%)**
	7	28	10	2002	*3.221 (-5.97%)*	3.785	**3.719 (-1.74%)**
	14	28	10	2786	*5.561 (-10.12%)*	6.420	**6.073 (-5.42%)**

two-layer network with 10 neurons on both layers.

Forecasting river flow turned out to be challenging for iterative method. The obtained results were much worse than baseline model, and worse than on networks developed by multi-neural networks. However, the error measures were consistent on validation and test sets.

Models developed by multi-neural network strategy had poor performance, as none of the models could improve over baseline model. The closest results to ARIMA model were obtained by small networks with a few neurons on the hidden layer. In addition, the relative error was lower in the case of longer forecasting horizon. Most of the models chosen as the best MLP models had good performance on both validation and test set.

Long Short-Term Memory Models with Package `Keras`

Overall, LSTM models had great performance, as it can be seen from Table 4. The improvement over baseline was succeeded with all models for forecasting horizon $H = 7$ or $H = 14$ having in best cases 6-7% lower error when compared to baseline model. The short forecasts produced errors close to ARIMA but were sometimes up to 10% worse than ARIMA. In all cases, the models that used more observations from the past had greater performance.

In case of short forecasting horizon, the networks with 100 LSTM cells had better forecasts, but in comparison, for forecasting long horizon, smaller LSTM networks produced lower error. All in all, differences were minimal, and therefore may not be significant.

On Figure 18 in Appendix F it can be seen the forecasts of best LSTM model for $H = 14$. The model obtained 5% lower error than chosen ARIMA model. From the figure it can be seen, that the model forecasts similar behavior of the true values. The great errors seem to occur when there is sudden increase of the river flow volume per second, but seems to capture the downfall movements.

3.4.4 Forecasting Electricity Prices in Estonian Power Market

This dataset has daily frequency, and it can contain seasonality, as discussed in Section 3.3.4. Taking this into account, the forecasting horizon H was chosen to be equal to 1, 3, 7, 14. Values for p, the number of previous observations, that were used for every horizon H were taken equal to 7, 14, 28, 42.

The analyzed dataset contained couple of extremely high values, that may strongly dictate the comparable error metrics. Thus, error measure for this dataset was chosen to be mean absolute value instead of sum of squares metric, similarly to Section 3.4.3. Error measure MAE was used for comparing developed models, but it was not used as cost functions when training the model. Lower errors were obtained overall by models trained with sum of squares cost function.

Additionally, a modified series was modeled. As the high peaks are outliars in the dataset, the corresponding values were modified. When the value was greater than 150% of preceding 7 days average price then the exceeding part was cut. However, the models trained with the modified data had similar behavior when compared to original data, and therefore, the results were not included in the analysis.

Both seasonal and regular naïve models were fitted and compared and in all cases seasonal naïve with seasonal period of 7 days gave better results. However, chosen ARIMA model outperformed seasonal naïve method.

The ARIMA model was not chosen with function `auto.arima`, instead a manual analysis was performed. As a result, a SARIMA$(3,0,0)\times(4,0,0)_7$ model was obtained. The model was chosen based on diagnostics which is given in Appendix B.

Overall, neural networks had good performance and many models improved over the SARIMA model. However, when SARIMA model needed longer history of past observations, neural networks had better performance with shorter input horizon.

The models developed with `neuralnet` package had problems of converging to a minima, and no models were obtained.

The best neural network models were developed with multi-neural network strategy. However, the best models of multi-neural network strategy did not obtain consistent level of error on both validation and test set. The LSTM models chosen based on validation set were consistently good on test set as well, and therefore are given below in Table 5. A comprehensive overview with the best models for used packages and modeling strategies is given in Appendix G.

Table 5. Comparison of the best neural network models and SARIMA forecasts for electricity prices.

Model			Hidden	Train	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{val})$	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{test})$	
type	H	p	neurons	time (s)	Neural network	SARIMA	Neural network
LSTM	1	28	100	7089	*3.610 (-1.96%)*	4.642	**4.450 (-4.15%)**
	3	14	100	4657	*4.265 (-0.46%)*	6.029	**5.810 (-3.63%)**
	7	14	20	2133	*4.368 (-1.57%)*	6.716	**6.136 (-8.64%)**
	14	14	10	699	*4.489 (-1.51%)*	**7.370**	7.753 (5.20%)

Multilayer Perceptrons with Package `h2o`

For every horizon H and number of input values p, the chosen hidden layers and hidden neurons were following: one-layer networks with $2, 3, \ldots, 9, 10, 20, 100$ neurons, and a two-layer network with 10 neurons on both layers.

The iterative strategy models performed well on this dataset when compared to other datasets. Although the obtained errors were higher than errors produced by multi-neural network models, the relative errors were not very high - never more than 38% higher error than baseline. Interestingly, many iterative models had better relative performance on test set when compared to validation set. This may be due to test period that had more irregular behavior when compared to validation set, causing SARIMA model to have high error measure on test set.

Models developed by multi-neural network strategy performed well, as many models were able to obtain lower error measure than SARIMA model. If the task was to obtain one-step-ahead forecast, it was enough to have 7 or 14 previous observations, but the models based on long history (p=42) worked well too. Also, in case of short forecasting horizon the models with few neurons performed better on test set, and large networks on validation set. However, both errors were lower than baseline, and the improvement was up to 10% lower MAE than SARIMA model.

When forecasting longer horizons, the larger networks had better performance on validation than small networks. However, the results may indicate slight overfitting, as the results on test set changed in the opposite direction - smaller networks had lower error when compared to large networks.

Long Short-Term Memory Models with Package `Keras`

The LSTM models for forecasting electricity prices did not obtain much better results when comparing with multi-neural network models. However, the models chosen on validation data proved to be consistently good on test set as well.

There were LSTM models that obtained results on validation set on the same level with the baseline model (relative error measures ranged from -2% to 3%), and models with slightly worse results, but no more than 10% of higher error when compared to baseline models. No distinctive patterns of chosen length of input and number of hidden neurons were discovered. In contrary, the results on test data were more extreme. Good performance was obtained for short forecasting horizon (H=1, or H=3), and the results were even better by favoring short input over long history, and small networks over large networks (7-10% lower error than with baseline model). In comparison, large networks for forecasting long horizon based on long history had 20-25% higher error than SARIMA model.

3.4.5 Forecasting New Orders in the Manufacturing Sector in the U.S.

As the dataset has monthly frequency, the forecasting horizon H for new orders dataset was chosen to be equal to 1, 3, 6, 12 to reflect forecasts for a month, whole quarter, half-year, or year. The series implies a seasonal component, and therefore, the value for p, the number of previous observations given as input, was chosen to be a multiple of $s = 12$, i.e., 12, 24, 36.

As explained in Section 3.3.5, the dataset is affected by the 2008 financial crisis, but values were left as they were, and no additional transformations were considered. This made it possible to demonstrate whether neural network models have consistent performance on the datasets with and without sudden changes in the behavior.

The extreme drop in new orders fell into period of validation set, and therefore it would be natural to assume that the validation errors are greater than errors on the test set. Depending on the number of preceding observations, it was the case. When the input forecast was shorter, i.e., 12 or 24 months, then the sudden drop was to be forecast, and therefore produced high errors. In contrary, when the input was longer, $p = 36$, then the sudden drop was already given as input, and it did not affect the errors in such scale.

The above mentioned behavior draw the attention how to compare different methods. This was great example for showing that when the methods are compared, the values that must be calculated **exactly** on the same data, and this includes also the linear models. In this way, the changes in behavior affect comparable results in the same magnitude.

Furthermore, this dataset demonstrated how inconsistent are the models chosen on basis of validation set results when validation set does not follow the same structure of the overall time series. To affirm the statement, there are four models given in Appendix H that had the best results on test set, and had great performance on validation set as well. This demonstrates that the neural network have potential to have great improvement over baseline model, if the model is not chosen based on the results on structurally different data.

For comparison of the methods, two linear baseline models were fitted. The SARIMA model obtained was SARIMA(0,1,1)$\times$(0,1,1)$_{12}$, where parameters were estimated based on preceding 36 observations. A naïve method was also used with seasonality, but lower errors on both validation and test set were obtained by the SARIMA model. Therefore, this was used as baseline for neural network method comparisons.

The best performing neural network models for all horizons H forecasting the sum of new orders in manufacturing sector are given below in Table 6, a comprehensive overview with the best models for every package and strategy is given in Appendix H.

Table 6. Comparison of the best performing multi-neural network strategy models with SARIMA forecasts for the new orders in manufacturing sector of the U.S.

Model			Hidden	Train	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{val})$	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{test})$	
type	H	p	neurons	time (s)	Neural network	SARIMA	Neural network
h2o: large	1	36	10,10	8	*10856 (-27.24%)*	**14199**	14416 (1.52%)
	3	36	200	59	*12375 (-33.99%)*	**14873**	16026 (7.76%)
	6	36	200	117	*15814 (-36.57%)*	**17591**	20830 (18.41%)
	12	36	300	231	*19980 (-37.32%)*	**22748**	24876 (9.35%)

Multilayer Perceptrons with Package `neuralnet`

The models developed with package `neuralnet` were one-layer networks with 3, 5, 8, 10,15,20 neurons in the hidden layer, and two-layer networks with 10 neurons on both layers. The results implied that there were no clear outlines as to which number of hidden neurons would help to describe the data in the best way.

Most of the obtained models performed better than SARIMA model on validation set, with few exceptions. The best NN models had up to 22% lower error when compared with baseline model. The highest improvements over baseline models were achieved by the models with longer forecasting horizon, e.g. 6 or 12 steps ahead.

Models chosen to be the best based on validation set did not persistently improve over baseline models when comparing the results on test set. In most of the cases, SARIMA model had better performance on test set, and some neural networks had even up to 50% higher error when compared to baseline.

Multilayer Perceptrons with Package `h2o`
For every horizon H and number of input values p, the chosen hidden layers and hidden neurons were following: one-layer networks with $3, 5, 10, 100, 200, 300$ neurons, and a two-layer network with 10 or 20 neurons on both layers.

Models developed by iterative strategy obtained results very poor performance on both validation and test set. This holds for both small and large networks. The relative error measures were up to two time higher on validation set and up to three times higher on test set when compared to baseline models. As a result of extremely poor performance compared to baseline, iterative models were not included for choosing the overall best model.

Models developed by multi-neural network strategy performed well by offering decent improvement over baseline models on validation set. Large networks obtained slightly lower error than smaller networks. This concurs with the fact that large networks are more flexible, and by that it may describe more behavioral patterns which could possibly indicate potential sudden changes, or help to smoothen the effect.

However, on test set the only improvement offered by neural networks was obtained when forecast horizon H was equal to 1. For longer horizons, the network models could not result in a lower error measure than SARIMA. Furthermore, none of the best models on validation set were had the lower error on the test set. As mentioned, there are four models that had the lowest error on test data shown in Appendix 3.3.5. These models serve the objective that there are models, that could have high improvement rate on both validation and test set. However, these models are often overlooked when the best model is chosen based on validation set that does not follow the structure of overall time series.

Long Short-Term Memory Models with Package `Keras`
Overall, LSTM models had inconsistent performance. Roughly half of the trained LSTM models did not show improvement over SARIMA model on the validation set, and all of the models had greater error measure on the test data.

More detailed analysis of the results revealed that the networks with 10 cells had similar performance behavior as SARIMA model. When the combination of forecast horizon H and number of inputs p was unfavorable, both of the models failed, and in case of favorable conditions, both models exceeded greatly the larger network models. The large networks had consistently second-rate performance throughout all combinations of H and p on both validation and test sets. Small LSTM model and SARIMA model performed remarkably better than larger networks on validation set when the number of past observations used was 12 or 36. In case of using 24 past observations, the results were on the same level with larger networks. This, again, can be considered as an example how comparing the models over a period with extreme events is affected greatly by the choice of parameters, and may result in inconsistent conclusions. It must be noted that on the test set which did not contain sudden changes in the behavior, small LSTM networks performed remarkably better.

As LSTM models demonstrated lack of consistency when the validation and test datasets were very different, the models were not included in the comparison for determining overall best neural network model. However, the results are still given in Appendix H.

3.5 Discussion

The analyzed datasets were chosen to contain at least some nonlinearities, demonstrate different structural behavior, and to show the potential of using neural networks for forecasting time series generated in different domains. As the datasets had different nature, the results exhibited many different behavioral patterns of the performance of neural network models. Furthermore, the packages used in the empirical study exposed how different data may affect the model training process and the performance of the models.

It must be noted that as all models trained were affected by random initialization of parameters, presence of multiple local minimas of the cost function, and therefore the results may vary in some extent when the networks are fitted with exactly the same attributes. However, the ensemble models were used specifically for the objective of lowering the variance of these models.

Also, it must be taken into account that the analysis was limited by time, and computational power, and it should be acknowledged that all of the results obtained could be improved by further actions. For example, by tuning further the hyperparameters, developing ensemble models with higher number of individual models, or by adding some additional information, e.g., day of the week, or some additional time series that has related behavior. These additional steps should be considered when the task is to find the best model for certain time series.

The conclusions of the work done in this thesis will be presented together with reasoning in the following sections. Additionally, two suggestions are given in Section 3.5.5 how the performance could be improved.

3.5.1 Domain and Structure of the Time Series

The EUR-USD exchange rate and Baltic Benchmark index datasets are from financial domain, and therefore are considered to follow random walk (Hyndman & Athanasopoulos, 2018). The best suitable models in these datasets had rather small number of neurons. When the number of neurons is small the network does not capture many patterns from the data. Thus, as small networks were mostly better than large networks, it may indicate that there are not many patterns in the observed data, and may contain strong random component.

The datasets with seasonality component, e.g. the electricity market prices and the new orders in manufacturing sector in US, offered many models that improved over SARIMA models. These datasets contained regular patterns that were great input for both LSTM and multi-neural networks models. However, in case of new orders dataset the selection of best models was greatly affected by the choice of validation set, and did not reflect the full potential of neural network models.

The river flow time series was challenging for multilayer perceptron models as the patterns were irregular and had high variance. However, LSTM models could benefit also from irregular patterns, and this may be the reason why these models had better results than MLP models, and furthermore offered improvement over baseline models.

3.5.2 Model Specifications

Model Architecture

The architecture with the overall best performance was long short-term memory model. The essence of LSTM models is to model sequential data. This is a desirable property when forecasting time series. In case of four datasets out of five, LSTMs were almost the only models that could offer improvement over chosen baseline models. This implies superiority of LSTM models.

However, long short-term memory models did not have great performance on the new orders dataset. The dataset was overall challenging for neural network models, as the validation set contained structural anomaly. Bad results on test set can be due to overfitting to the validation set. However, as the error was very high even on validation set in case of forecasting 1 and 3 steps ahead, there must be another source for unsatisfactory results. It is very possible that the observed dataset had too few observations, and the LSTM just could not find good solutions based on given data.

The multilayer perceptrons can be used for forecasting time series, as the results indicate that the forecasts have some predictive power. However, the models obtained in this thesis were mostly not able to improve over baseline models. As the linear models are easily interpretable, and can be expressed analytically, these models are preferred for time series analysis, unless the performance of using multilayer perceptrons is improved.

The Modeling Strategy

When comparing the modeling strategies, multi-neural networks developed with `h2o` and models with direct approach using `neuralnet` had competitive performance when compared to baseline models (when the optimization algorithm converged).

The iterative method had the worst performance on each of the dataset. However, as further analysis revealed, MAPE error measure was less than 10% in all cases, indicating presence of predictive power for this method also. Furthermore, it is natural that if the obtained forecasts are used as input for obtaining new forecasts, the errors are magnified with each reiteration. One possibility to overcome this shortcoming is to use much bigger ensemble model. As the variance of iterative models is assumed to be higher, this could be lowered when aggregating over high number of individual models.

The Size of Networks

Networks used in time series forecasting do not need to have necessarily high number of neurons and hidden layers. It depends on the nature of the dataset. If the dataset contains strong patterns, as in dataset of new orders in manufacturing sector, it is logical that the networks should need higher number of neurons to capture as much patterns as possible. When the data is considered to have strong random component, then it may be enough to use only small NNs, as for the EUR-USD exchange rate series.

3.5.3 Combinations of input and forecasting horizon

Different datasets had different patterns for the forecasting horizon. For example, the one-step-ahead forecasts of electricity market prices offered improvement over baseline models in case of all types of models. However, the errors were relatively much higher when comparing with the baseline.

In case of electricity prices, the best neural networks required a small number of inputs. SARIMA implied based on order $P = 4$, however, that the input should be higher number. The disagreement could be caused by the inconsistency of neural network models, and may be result of inconsistent conclusion over needed input. Also, it would be reasonable to include only the previous observations determined as informative by SARIMA model. However, ideally the a large enough neural network with proper regularization terms should be able to detect the informative previous variables itself, and therefore this approach was not tested in this empirical study.

With the Baltic Benchmark index and EUR-USD exchange rate data the shorter forecast has much better relative errors. In comparison, it can not be concluded for river flow and new orders forecasting.

Interestingly, in case of river flow forecasting, the small networks preferred less input, and large network preferred more input. This resulted in consistent models when comparing validation and test set. However, the performance overall was not great, and therefore the models were not highlighted.

3.5.4 Different Packages for Modeling

Both `Keras` and `h2o` package offered advanced optimization algorithms, effective multithreaded calculations with high flexibility in hyperparameter tuning. Additionally, with `Keras`, it is possible to develop networks with very flexible and customized structure. Despite the great performance, results for LSTM models come with the cost of computing power and time spent. However, when using the models for future forecasts, the model is already trained and the time spent on prediction is much lower. The trade-off between performance and cost should be chosen for each specific task for given risk tolerance.

The package `neuralnet` seems to be outdated. It can be useful in simple neural network training, but does not offer flexibility and effective algorithms for more complex tasks. The method tkas long time to converge. In case of three analyzed datasets the training algorithm did not converge within allowed iteration steps. The possibilities could be difficult patterns, or strong random component.

3.5.5 Future Work

The neural network models will always benefit from larger training set. However, in case of time series the length of data is limited by the frequency of observations. The neural network models fitted in the empirical study had mostly daily frequency. As there were not very many good performing models, it may indicate that the random component is difficult to forecast. By this reasoning, a possible solution could be predicting smoothed series. By using smoothed data, it still has high frequency, but now the model could focus more on the behavioral changes instead of random changes.

The hybrid models introduced in Section 2.5.3 could offer the remedy by combining reliable linear models with neural networks to obtain a hybrid forecast.

Conclusion

The first objective of this thesis was to give theoretical guidance on applying neural network models on time series forecasting task. In addition, an empirical study was conducted in order to give practical guidance to develop neural network models for forecasting univariate time series with packages `neuralnet` and `h2o` for `R`, and `Keras` for `Python`. The performance of the models was compared based on different forecasting horizons, the number of preceding observations, different modeling strategies, neural network architectures and network structure. To assess whether the obtained models have predictive power, they were compared to chosen baseline models.

In the empirical study multilayer perceptron and long short-term memory models were fitted on five datasets containing different nonlinear behavior. The models were constructed with different modeling approaches – iterative, direct and multi-neural network strategy. For each dataset there were multiple different forecasting horizons modeled, to show if the performance is affected by the horizon H. Additionally, for developing models, grid search was used to iterate over different number of input observations and number of hidden neurons and layers.

In four datasets out of five the LSTM models demonstrated the lowest errors when compared to baseline models and as well to multilayer perceptron models. Unfortunately, the performance of MLP models mostly did not show improvement over linear baseline models. The reasons for underperformance can be strong random component, irregular patterns, or structural changes in data.

Financial time series that can be considered to follow random walk, the neural network models that had better performance contained small number of neurons. This could be due to lack of patterns present in the training data that could be captured by neural networks.

Forecasting river flow of Great Emajõgi was too complex task for multilayer perceptrons, at least for chosen hyperparameter space. In comparison, the series of electricity market prices was suitable for multilayer perceptrons, as many models achieved improvement over baseline models. However, the models often did not have consistent performance, and the models chosen as the best based on validation set performed poorly on test set.

Time series consisting of new orders in manufacturing sector in US that suffered from 2008 financial crisis was analyzed as an example to highlight the importance of using proper validation set. It demonstrated that when the validation set does not have the same behavioral structure as the overall time series, the models chose based on validation set results will not give consistent results.

When comparing the different modeling strategies, the models developed by iterative approach resulted in much higher error measures. This indicates the superiority of direct and multi-neural network strategies.

All in all, the thesis served the purpose of giving both theoretical and practical guidelines for using neural networks in time series forecasting. Hereafter, when it is needed to find neural network model for a specific time series, this thesis can be used as a starting point.

References

Adhikari, R., Agrawal, R. K. (2013). A combination of artificial neural network and random walk models for financial time series forecasting. *Neural Computing and Applications*, 24(6), 1441–1449. doi: `https://doi.org/10.1007/S00521-013-1386-Y`

Allaire, J. J., Chollet, F. (2019) *R interface to Keras.* Retrieved on May 10 , 2019 from `https://keras.rstudio.com/index.html`

Ben Taieb, S., Bontempi, G., Atiya, A., Sorjamaa, A. (2012) A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. *Expert Systems with Applications.* 39-8: 7067-7083 (2012). doi: `https://doi.org/10.1016/j.eswa.2012.01.039`

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford: Clarendon Press. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.679.1104&rep=rep1&type=pdf`

Boehmke, B. (2018) *Feedforward Deep Learning Models* Retrieved on April 27, 2019, from `http://uc-r.github.io/feedforward_DNN`.

Brockwell, P. J., Davis, R. A. (2002). *Introduction to Time Series and Forecasting.* (Second edition). New York: Springer-Verlag.

Brownlee, J. (2018) *How to Develop Multilayer Perceptron Models for Time Series Forecasting* Retrieved on April 28, 2019, from
`https://machinelearningmastery.com/how-to-develop-multilayer-perceptron-\models-for-time-series-forecasting/`.

Candel, A., LeDell, E. (2019). *Deep Learning with H2O.* Retrieved on May 8, 2019, from `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/DeepLearningBooklet.pdf`.

Carillo, M., González, J. A., Gracia-Linares, M. (2015). Time Series Analysis of Gravitational Wave Signals Using Neural Networks. *Journal of Physics: Conference Series*, Volume 654. doi:`https://doi.org/10.1088/1742-6596/654/1/0120010`

Changhau, I. (2017). *Loss Functions in Neural Networks.* Retrieved on April 15, 2019, from `https://isaacchanghau.github.io/post/loss_functions/`.

Chollet, F., et al. (2015). *Keras.* Retrieved on May 10, 2019 from `https://keras.io`

Estonian Weather Service. (2019). Internal Waters. Historical Data. Retrieved on April 18, 2019, from `http://www.ilmateenistus.ee/siseveed/ajaloolised-vaatlusandmed/`

Fritsch, S., Guenther, F., Wright, M. N. (2019). *Package 'neuralnet'* Retrieved on May 5, 2019, from `https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf`. R package version 1.44.2.

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning.* MIT Press.
`http://www.deeplearningbook.org`.

Haykin, S. (2009). *Neural Networks and Learning Machines* (3rd Edition). New Jersey: Pearson Education, Inc.

Hebb, D. (1949). *The Organization of Behavior.* New York: Wiley.

Hinton, G. (2006). *To Recognize Shapes, First Learn to Generate Images.* Technical Report UTML TR 2006-003, University of Toronto.
`http://www.cs.toronto.edu/~fritz/absps/montrealTR.pdf`.

Hochreiter, S., Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computations* 9(8): 1735-1780, 1997. `http://www.bioinf.jku.at/publications/older/2604.pdf`

Hyndman, R. J., Athanasopoulos, G. (2018). *Forecasting: Principles and Practice.* (Second edition). Melbourne, OTexts.
Retrieved April 20, 2019, from `https://otexts.com/fpp2/`.

Hyndman, R. J., Ben Taieb, S. (2012). *Recursive and direct multi-step forecasting: the best of both worlds.* `https://robjhyndman.com/papers/rectify.pdf`.

Hyndman, R. J., Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting,* 22(4): 679-688, 2006. `https://robjhyndman.com/papers/mase.pdf`.

Kourentzes, N., Barrow, D. K., Crone, S. F. (2014). Neural Neural Network Ensemble Operators for Time Series Forecasting. *Expert Systems with Applications,* 41: 4235-4244, 2017. doi: `https://doi.org/10.1016/j.eswa.2013.12.011`

Kuan, C.-M., Liu, T. (1995). Forecasting Exchange Rates Using Feedforward and Recurrent Neural Networks. *Journal of Applied Econometrics*, 1995, vol. 10, issue 4, 347-64. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.8265&rep=rep1&type=pdf`

Lapedes, A., Farber, R. (1987). *Nonlinear signal processing using neural networks: Prediction and system modelling.* United States.

LeDell, E., Gill, N., Aiello, S., Fu, A., Candel, A., Click, C., . . . , Malohlava, M. (2019) *Package 'h2o'* Retrieved on May 5, 2019, from `https://cran.r-project.org/web/packages/h2o/h2o.pdf`. R package version 3.22.1.1.

Lee, S., Jeong, T. (2017) Forecasting Purpose Data Analysis and Methodology. Comparison of Neural Model Perspective. *Symmetry*, 9(7): 108 (2017). doi: `https://doi.org/10.3390/sym9070108`

Lin, T., Guo, T., Aberer, K. (2017). Hybrid Neural Networks for Learning the Trend in Time Series. *Proceedings of the 26th International Joint Conference on Artificial Intelligence,* 2273-2279. Melbourne, AAAI Press.

Macrotrends LLC. (2019). Euro Dollar Exchange Rate (EUR USD) - Historical Chart. Retrieved on March 13, 2019, from `https://www.macrotrends.net/2548/euro-dollar-exchange-rate-historical-chart`

Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., . . . , Winkler, R. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting* 1(2): 111-153, 1982. `https://doi-org.ezproxy.utlib.ut.ee/10.1002/for.3980010202`

Makridakis, S., Spiliotis, E., Assimakopoulos, V. (2018). Statistical and Machine Learning forecasting methods: Concerns and ways forward. *PLoS ONE* 13(3): e0194889. https://doi.org/10.1371/journal.pone.0194889

Maqsood, I., Khan, M.R., Abraham, A. (2004) An ensemble of neural networks for weather forecasting. *Neural Computing & Applications*, 13: 112 (2004). doi: `https://doi-org.ezproxy.utlib.ut.ee/10.1007/s00521-004-0413-4`

Minsky, M., Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry.*

MIT Press

Montaño Moreno, J. J., Pol, A. P., Gracia, P. M. (2011). Artificial neural networks applied to forecasting time series. *Psicothema*, 23(2): 322-329 (2011).

Nasdaq Baltic. (n.d.-a). *About Indexes.* Retrieved on May 10, 2019 from `https://nasdaqbaltic.com/market/?pg=charts&lang=en`

Nasdaq Baltic. (n.d.-b). *Baltic Market Indexes.* Retrieved on March 23, 2019 from `https://nasdaqbaltic.com/en/about-indexes/`

Nasdaq Baltic. (2016). *Rules for the Construction and Maintenance of the Nasdaq OMX Baltic Equity Indexes.* Retrieved on May 10, 2019 from `https://www.nasdaqbaltic.com/files/tallinn/oigusaktid/Indeks/Methodology_OMXBALTIC_2016.pdf`

Nielsen, M. A. (2015). *Neural Networks and Deep Learning.* Determination Press. `http://neuralnetworksanddeeplearning.com/`

Nord Pool AS. (2019). Historical Market Data. Retrieved on April 28, 2019, from `https://www.nordpoolgroup.com/historical-market-data/`

Olah, C. (2015). *Understanding LSTM Networks.* Retrieved on April 27, 2019, from `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Ord, K., Fildes, R., Kourentzes, N. (2017). *Principles of business forecasting.* 2nd edition. New York: Wessex Press Publishing Co.

Rosenblatt, F. (1958). *The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain.* Psychological Review. 65(6): 386–408. doi:10.1037/h0042519.

Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge. `https://academiaanalitica.files.wordpress.com/2016/11/david-e-rumelhart-james-l-mcclelland-pdp-research-group-parallel-distributed-\processing_-explorations-in-the-microstructure-of-cognition_-foundations-vol-11986.pdf`.

Siami-Namin, S., Siami Namin, A. (2018). *Forecasting Economic and Financial Time Series: ARIMA vs. LSTM.* https://arxiv.org/abs/1803.06386v1

Singh, R., Balasundaram, S. (2007). Application of Extreme Learning Machine Method for Time Series Analysis. *International Journal of Intelligent Systems and Technologies*, 2(4).

Srinivasan, D., Liew, A. C., Chang, C. S. (1994). A neural network short-term load forecaster. *Electric Power Systems Research,* 28: 227-234. doi: `https://doi.org/10.1016/0378-7796(94)90037-X`

Tashman, L. J. (2000). Out-of sample tests of forecasting accuracy: an analysis and review. *International Journal of Forecasting,* 16: 437-450, 2000.

U.S. Census Bureau. (2018). Manufacturers' Shipments, Inventories, & Orders. Historical Data. Retrieved on April 18, 2019, from `https://www.census.gov/manufacturing/m3/historical_data/index.html`

White, H. (1988). *Economic prediction using neural networks: the case of IBM daily stock returns.* IEEE 1988 International Conference on Neural Networks, San Diego, CA, USA, 1988, vol.2. doi: 10.1109/ICNN.1988.23959

Zhang, G. (1994). Time series analysis and prediction by neural networks. *Optimization Methods and Software,* 4: 151-170, 1994. doi: `https://doi.org/10.1016/S0925-2312(01)`

00702-0

Zhang, G. P. (2002). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing,* 50: 159-175, 2003.

Zhang, G., Patuwo, B. E., Hu, M. Y. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting* 14: 35-62, 1998.

Appendixes

A Appendix - Algorithms

Algorithm 1: The back-propagation algorithm for fully connected MLP. Based on Algorithm 6.3 and Algorithm 6.4 presented by Goodfellow et al. in their book (2016). The operator $\odot$ implies element-wise multiplication. The output is fed into optimization algorithm, such as stochastic gradient decent introduced in Section 1.4.2.

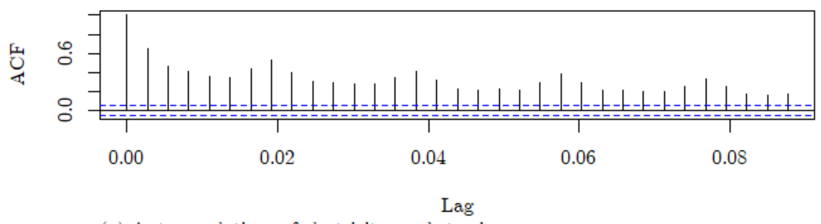
```
1. Forward propagation
Inputs:
  • $l$, the depth of the network
  • $\mathbf{W}^{(i)}, i \in \{1,2,\ldots,l\}$, the weight matrices for each layer;
  • $\mathbf{b}^{(i)}, i \in \{1,2,\ldots,l\}$, the bias parameters for each layer;
  • $\mathbf{x}$, the input variables of the observation;
  • $\mathbf{y}$, the true output values.
$\hat{\mathbf{y}}^{(0)} = \mathbf{x}$
for $k = 1,\ldots,l$ do
    $\mathbf{v}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\hat{\mathbf{y}}^{(k-1)}$
    $\hat{\mathbf{y}}^{(k)} = \varphi(\mathbf{v}^{(k)})$
end
$E = \mathcal{E}(\hat{\mathbf{y}}^{(k)}, \mathbf{y})$

2. Backwards propagation
Compute the gradient on the output layer:
$\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}^{(l)}} E$
for $k = l,l-1,\ldots,1$ do
    Convert the output gradient into a net input gradient:
    $\mathbf{g} \leftarrow \nabla_{\mathbf{v}^{(k)}} E = \mathbf{g} \odot \varphi'(\mathbf{v}^{(k)})$
    Compute gradients for parameters:
    $\nabla_{\mathbf{b}^{(k)}} E = \mathbf{g}$
    $\nabla_{\mathbf{W}^{(k)}} E = \mathbf{g}\hat{\mathbf{y}}^{(k-1)T}$
    Compute the gradient on the output of preceding hidden layer:
    $\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}^{(k-1)}} E = \mathbf{W}^{(k)T}\mathbf{g}$
end
Outputs:
  • $\nabla_{\mathbf{b}^{(k)}} E, k \in \{1,2,\ldots,l\}$, gradient on bias terms for each layer;
  • $\nabla_{\mathbf{W}^{(k)}} E, k \in \{1,2,\ldots,l\}$, gradient matrix on weights for each layer.
```

Algorithm 2: The stochastic gradient descent algorithm. Based on Algorithm 8.2 presented by Goodfellow et al. in their book (2016). If $\alpha > 0$, the algorithm corresponds to SGD with momentum.

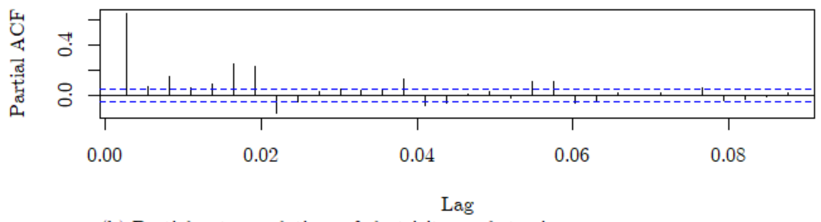
```
Inputs:
  • $\eta$, learning rate;
  • $\alpha$, the parameter for momentum term;
  • $\boldsymbol{\theta}$, initial values for parameters, biases and weights;
  • $\mathbf{m}$, the initial value of weighted average of gradients;
while stopping criterion not met do
    Draw a random sample of $n$ observations $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$ with target values $\mathbf{y}^{(i)}$ from the entire training set.
    Compute the estimate for gradient: $\mathbf{g} \leftarrow \frac{1}{n}\nabla_{\boldsymbol{\theta}} \sum_i \mathcal{E}(\hat{\mathbf{y}}^{(i)},\mathbf{y}^{(i)})$.
    Compute momentum update: $\mathbf{m} \leftarrow \alpha\mathbf{m} - \eta\mathbf{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$.
end
```

B Appendix - Diagnostics of SARIMA Model For Electricity Market Prices

SARIMA(3,0,0)$\times$(4,0,0)$_7$ was the model for electricity market prices in Estonia. The determination of SARIMA orders was based on Figure 15 and Figure 16.



(a) Autocorrelations of electricity market prices.



(b) Partial autocorrelations of electricity market prices.

Figure 15. Auto and partial autocorrelations of electricity market price series

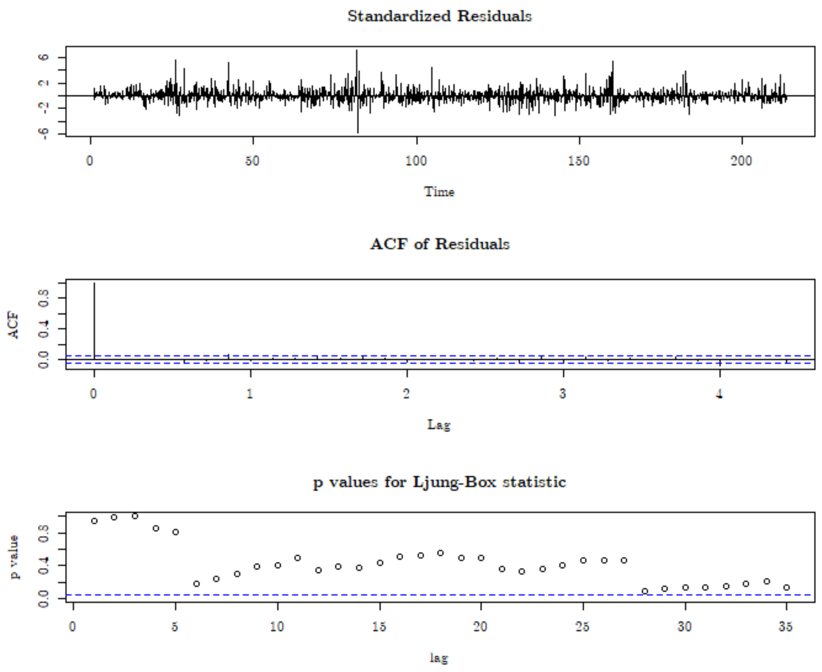
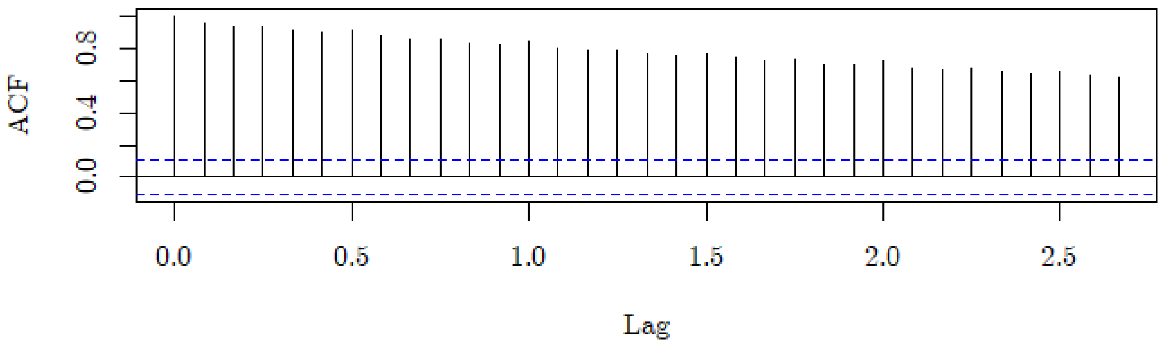


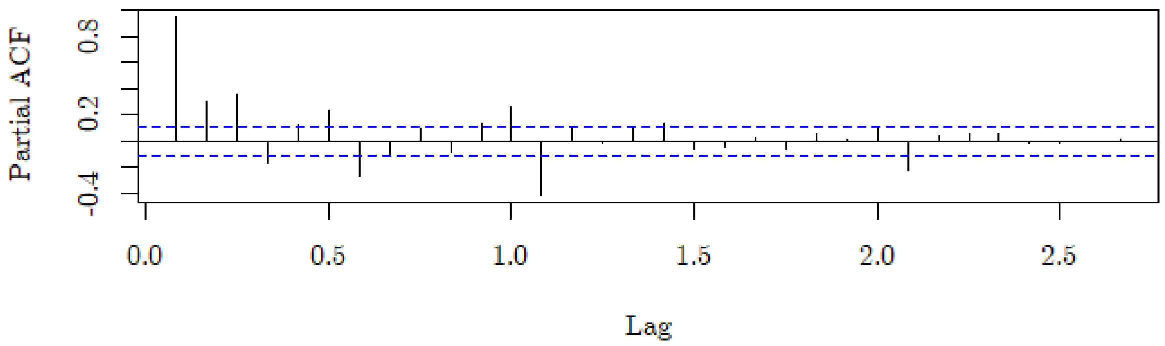
Figure 16. Diagnostics of SARIMA(3,0,0)$\times$(4,0,0)$_7$ model residuals.

C Appendix - Diagnostics of the New Orders in Manufacturing Sector

The following Figure 17 describes auto- and partial autocorrelations in the series of new orders in manufacturing sector.



(a) Autocorrelations of new orders in manufacturing sector.



(b) Partial autocorrelations of new orders in manufacturing sector.

Figure 17. Auto and partial autocorrelations of new orders in manufacturing sector.

D Appendix - Results of Forecasting EUR-USD Exchange Rate

Comparison of the best performing neural network models with naïve forecasts for EUR-USD exchange rate.

Model			Hidden	Train	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{val})$	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{test})$		Test
type	H	p	neurons	time (s)	Neural network	Naïve	Neural network	rank
LSTM	1	2	20	604	*0.00548 (-17.90%)*	0.00485	**0.00417 (-14.00%)**	1/6
	3	6	20	603	*0.00842 (-6.88%)*	0.00652	**0.00640 (-1.74%)**	1/6
	5	10	10	878	*0.01025 (-5.64%)*	0.00783	**0.00776 (-0.92%)**	1/5
	10	50	10	7512	*0.01361 (-6.81%)*	0.01040	**0.01010 (-2.92%)**	1/6
	15	30	10	1623	*0.01539 (-11.41%)*	0.01238	**0.01194 (-3.57%)**	2/6
neuralnet	1	5	3	1062	*0.00668 (0.10%)*	0.00485	**0.00484 (-0.16%)**	1/24
	3	6	6	7433	*0.00916 (1.23%)*	**0.00652**	0.00656 (0.70%)	1/2
h2o: small*	1	2	3	12	*0.00672 (0.81%)*	**0.00485**	0.00485 (0.11%)	1/24
	3	6	8	61	*0.00919 (1.59%)*	**0.00652**	0.00668 (2.49%)	1/24
	5	10	3	92	*0.01119 (2.99%)*	**0.00783**	0.00804 (2.65%)	1/24
	10	20	4	194	*0.01526 (4.59%)*	**0.01039**	0.01083 (4.26%)	1/27
	15	30	3	307	*0.01815 (4.50%)*	**0.01238**	0.01319 (6.57%)	1/11
h2o: big*	1	2	100	21	*0.00676 (1.28%)*	**0.00485**	0.00488 (0.73%)	1/12
	3	6	50	66	*0.00931 (2.96%)*	**0.00652**	0.00671 (3.02%)	1/12
	5	10	20	94	*0.01118 (2.91%)*	**0.00783**	0.00830 (6.01%)	3/12
	10	20	100	323	*0.01604 (10.00%)*	**0.01039**	0.01161 (11.68%)	1/8
	15	30	20	313	*0.01868 (7.55%)*	**0.01238**	0.01364 (10.13%)	1/4
h2o: small**	1	2	3	20	*0.00678 (1.70%)*	**0.00485**	0.00491 (1.31%)	2/24
	3	6	7	120	*0.00916 (1.31%)*	**0.00652**	0.00775 (18.93%)	1/24
	5	10	4	234	*0.01092 (0.51%)*	**0.00783**	0.01038 (32.59%)	7/24
	10	20	8	504	*0.01457 (-0.14%)*	**0.01039**	0.01445 (39.04%)	5/8
	15	30	8	904	*0.01851 (6.56%)*	**0.01238**	0.01672 (35.03%)	3/11
h2o: big**	1	2	20	24	*0.00671 (0.61%)*	**0.00485**	0.00486 (0.33%)	1/12
	3	15	20	137	*0.00935 (3.24%)*	**0.00654**	0.00974 (49.03%)	9/12
	5	10	10,10	232	*0.0112 (3.06%)*	**0.00783**	0.01047 (33.78%)	4/12
	10	20	20	604	*0.01606 (10.11%)*	**0.01039**	0.01479 (42.36%)	1/3
	15	30	20	919	*0.01791 (3.09%)*	**0.01238**	0.01673 (35.10%)	1/4

* The models were developed by multi-neural network strategy.

** The models were developed by iterative strategy.

E Appendix - Results of Forecasting OMX Baltic Benchmark Index

Comparison of the best performing neural network models with ARIMA(1,1,0) forecasts for OMX Baltic Benchmark index.

Model			Hidden	Train	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{val})$	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{test})$		Test
type	H	p	neurons	time (s)	Neural network	ARIMA	Neural network	rank
LSTM	1	10	10	1189	*3.188 (-2.11%)*	4.494	**4.414 (-1.76%)**	1/9
	3	15	20	1417	*4.405 (-5.17%)*	6.233	**6.147 (-1.37%)**	2/9
	5	25	10	1977	5.426 (-7.65%)	7.738	**7.728 (-0.12%)**	7/9
	10	20	10	643	7.739 (-11.27%)	**10.640**	10.779 (1.30%)	5/6
	15	30	20	1050	9.990 (-13.16%)	**13.001**	13.191 (1.46%)	1/5
h2o: small*	1	2	5	10	*3.181 (-2.32%)*	**4.463**	4.467 (0.10%)	3/27
	3	30	3	45	*4.417 (-5.23%)*	**6.301**	6.340 (0.61%)	13/27
	5	25	4	96	*5.415 (-7.84%)*	**7.738**	7.884 (1.89%)	23/27
	10	20	7	201	*7.574 (-13.16%)*	**10.640**	10.934 (2.76%)	9/25
	15	30	9	308	*9.595 (-16.59%)*	**13.001**	13.552 (4.24%)	3/27
h2o: large*	1	2	50	19	*3.205 (-1.58%)*	**4.463**	4.491 (0.64%)	4/6
	3	6	100	62	*4.454 (-3.89%)*	**6.200**	6.258 (0.94%)	1/6
	5	10	100	102	*5.547 (-5.39%)*	**7.667**	7.759 (1.20%)	2/6
	10	20	50	202	*7.808 (-10.48%)*	**10.640**	11.155 (4.84%)	3/6
	15	30	50	320	*9.890 (-14.03%)*	**13.001**	13.570 (4.38%)	1/6
h2o: small**	1	2	6	12	*3.203 (-1.66%)*	**4.463**	4.467 (0.09%)	3/27
	3	15	4	112	*4.380 (-5.71%)*	**6.233**	6.493 (4.17%)	4/26
	5	25	7	260	*5.361 (-8.77%)*	**7.738**	8.296 (7.21%)	12/27
	10	20	5	502	*7.521 (-13.77%)*	**10.640**	11.847 (11.34%)	2/10
h2o: large**	1	2	50	23	*3.189 (-2.09%)*	**4.463**	4.482 (0.44%)	3/6
	3	6	50	129	*4.393 (-5.22%)*	**6.200**	7.422 (19.70%)	3/6
	5	10	100	288	*5.469 (-6.72%)*	**7.667**	8.627 (12.52%)	3/6
	10	20	50	745	*7.915 (-9.25%)*	**10.640**	12.028 (13.04%)	1/2

* The models were developed by multi-neural network strategy.
** The models were developed by iterative strategy.

F Appendix - Results of Forecasting River Flow of The Great Emajõgi

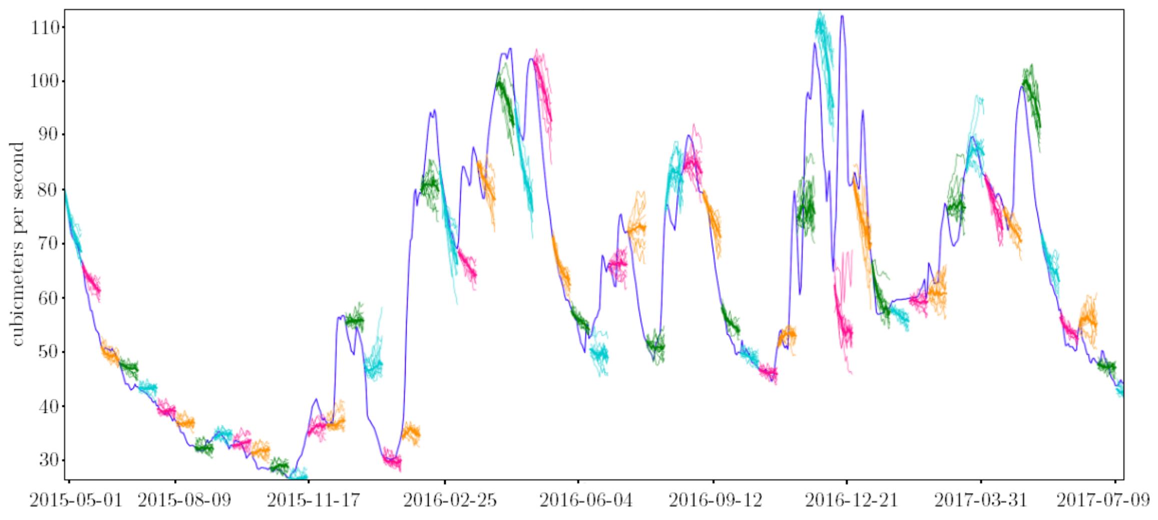
Comparison of the best performing neural network models with ARIMA(1,1,2) forecasts for the river flow of the Great Emajõgi.

Table 7. Comparison of the best performing neural network models with ARIMA(1,1,2) forecasts for the river flow of the Great Emajõgi.

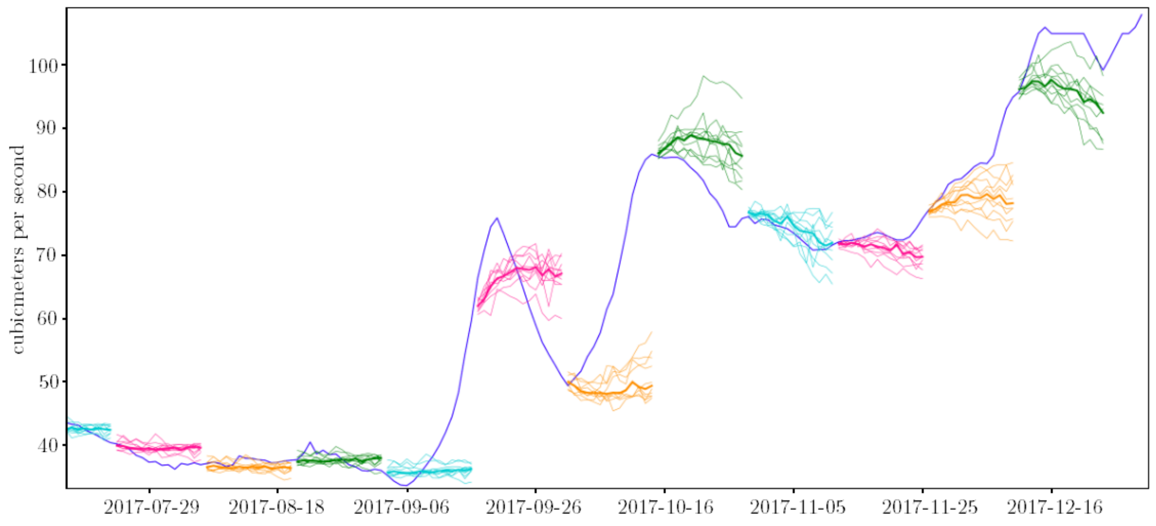
Model			Hidden	Train	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{val})$	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{test})$		Test
type	H	p	neurons	time (s)	Neural network	ARIMA	Neural network	rank
LSTM	1	35	100	1077	***0.750 (-1.04%)***	0.881	**0.880 (-0.08%)**	2/12
	3	35	100	907	***1.588 (-3.85%)***	1.889	**1.869 (-1.07%)**	1/12
	7	28	10	2002	***3.221 (-5.97%)***	3.785	**3.719 (-1.74%)**	5/12
	14	28	10	2786	***5.561 (-10.12%)***	6.420	**6.073 (-5.42%)**	6/12
h2o: small*	1	7	2	23	*0.808 (5.91%)*	**0.881**	0.942 (7.12%)	2/45
	3	7	3	64	*1.760 (5.92%)*	**1.887**	1.999 (5.91%)	4/45
	7	7	4	156	*3.593 (4.22%)*	**3.793**	3.879 (2.28%)	2/10
	14	28	2	325	*6.125 (-1.01%)*	**6.420**	6.601 (2.82%)	15/45
h2o: large*	1	14	100	25	*0.915 (20.25%)*	**0.879**	0.990 (12.60%)	3/15
	3	28	200	132	*2.001 (20.96%)*	**1.883**	2.158 (14.64%)	2/15
	7	7	100	234	*3.784 (9.74%)*	**3.793**	3.959 (4.37%)	2/3
	14	14	200	488	*6.724 (8.44%)*	**6.448**	6.693 (3.81%)	2/14
h2o: small**	1	7	7	27	*0.797 (4.46%)*	**0.880**	0.913 (3.85%)	1/45
	3	7	2	147	*2.294 (38.01%)*	**1.887**	3.932 (108.33%)	9/44
	7	7	7	434	*6.096 (76.79%)*	**3.793**	6.164 (62.52%)	5/9
h2o: large**	1	28	100	45	*1.027 (35.31%)*	**0.878**	0.996 (13.44%)	1/15
	3	14	200	287	*2.983 (80.13%)*	**1.887**	4.360 (131.02%)	14/15
	7	7	200	934	*7.307 (111.91%)*	**3.793**	6.310 (66.37%)	2/3

* The models were developed by multi-neural network strategy.

** The models were developed by iterative strategy.



(a) LSTM model forecasts in time period May 1, 2015 to July 9, 2017.



(b) LSTM model forecasts in time period July 10, 2017 to December 31, 2017.

Figure 18. LSTM(10) forecasts of river flow of the Great Emajõgi for horizon $H = 14$ based on 28 preceding observations. The lines show individual forecasts of the ensemble. Median forecast is marked with bold line. For graphical reasons, only forecasts for every 14 steps are depicted.

G Appendix - Results of Forecasting Electricity Market Prices

Comparison of the best performing neural network models with SARIMA(3,0,0)×(4,0,0)$_7$ forecasts for electricity market prices in Estonia.

Model type	H	p	Hidden neurons	Train time (s)	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{val})$ Neural network	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{test})$ SARIMA	$\overline{\text{MAE}}(\hat{\mathbf{Y}}^{test})$ Neural network	Test rank
LSTM	1	28	100	7089	*3.610 (-1.96%)*	4.642	**4.450 (-4.15%)**	5/8
	3	14	100	4657	*4.265 (-0.46%)*	6.029	**5.810 (-3.63%)**	3/7
	7	14	20	2133	*4.368 (-1.57%)*	6.716	**6.136 (-8.64%)**	2/7
	14	14	10	699	*4.489 (-1.51%)*	7.370	**7.753 (5.20%)**	2/7
h2o: small*	1	7	9	18	*3.711 (-8.17%)*	5.034	**4.794 (-4.76%)**	18/45
	3	7	10	58	*4.108 (-8.55%)*	6.262	**5.875 (-6.17%)**	11/45
	7	7	9	127	*4.236 (-7.95%)*	**6.773**	7.039 (3.93%)	42/45
	14	14	9	271	*4.209 (-7.64%)*	**7.370**	7.687 (4.31%)	9/13
h2o: small**	1	7	9	23	*3.692 (-8.62%)*	5.034	**4.758 (-5.47%)**	16/45
	3	7	8	119	*4.663 (3.79%)*	**6.262**	6.271 (0.15%)	12/45
	7	7	5	296	*4.358 (-5.30%)*	**6.773**	7.576 (11.86%)	38/44
	14	14	8	787	*4.508* (-1.08%)	7.370	**7.025 (-4.68%)**	3/10
h2o: large*	1	7	10,10	20	*3.701 (-8.41%)*	5.034	**4.872 (-3.22%)**	4/15
	3	14	10,10	59	*3.879 (-9.48%)*	**6.029**	6.125 (1.60%)	5/15
	7	7	100	154	*4.147 (-9.89%)*	**6.773**	8.403 (24.08%)	14/15
	14	14	10,10	367	*4.179 (-8.30%)*	**7.370**	8.348 (13.27%)	2/3
h2o: large**	1	7	100	26	*3.691 (-8.65%)*	**5.034**	5.195 (3.20%)	8/15
	3	7	10,10	130	*4.819 (7.27%)*	**6.262**	6.516 (4.06%)	8/15
	7	7	10,10	342	*4.729 (2.75%)*	**6.773**	7.419 (9.54%)	11/15
	14	14	20	1287	*4.693 (2.97%)*	**7.370**	8.471 (14.93%)	1/3

* The models were developed by multi-neural network strategy.

** The models were developed by iterative strategy.

H Appendix - Results of Forecasting Orders in Manufacturing Sector

Comparison of the best performing neural network models with SARIMA(0,1,1)$\times$0,1,1)$_{12}$ forecasts for the sum of new orders in manufacturing sector.

Model			Hidden	Train	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{val})$	$\overline{\text{RMSE}}(\hat{\mathbf{Y}}^{test})$		Test
type	H	p	neurons	time (s)	Neural network	SARIMA	Neural network	rank
h2o: large*	1	36	10,10	8	*10856 (-27.24%)*	**14199**	14416 (1.52%)	9/15
	3	36	200	59	*12375 (-33.99%)*	**14873**	16026 (7.76%)	7/15
	6	36	200	117	*15814 (-36.57%)*	**17591**	20830 (18.41%)	11/15
	12	36	300	231	*19980 (-37.32%)*	**22748**	24876 (9.35%)	5/15
h2o: small*	1	36	10	7	*11726 (-21.40%)*	14199	**13862 (-2.38%)**	5/12
	3	36	8	24	*13145 (-29.88%)*	**14873**	16745 (12.59%)	8/12
	6	36	10	51	*16826 (-32.51%)*	**17591**	19842 (12.79%)	4/12
	12	36	10	83	*21833 (-31.51%)*	**22748**	25109 (10.38%)	6/12
neuralnet	1	36	15	34	*11984 (-19.68%)*	**14199**	21994 (54.89%)	17/18
	3	36	10	446	*15038 (-19.79%)*	**14873**	19191 (29.03%)	11/13
	6	36	20	1601	*20827 (-16.47%)*	**17591**	24021 (36.55%)	8/10
	12	36	5	873	*26715 (-16.20%)*	**22748**	23694 (4.16%)	1/10
LSTM	1	36	10	2034	*22981 (54.03%)*	**14199**	27762 (95.52%)	3/9
	3	36	10	2735	*18756 (0.05%)*	**14873**	20443 (37.45%)	3/9
	6	24	20	1767	*31086 (-17.87%)*	**18765**	36730 (95.73%)	6/9
	12	24	20	2430	*27711 (-43.02%)*	**21810**	31756 (45.60%)	6/9
h2o: small**	1	36	5	14	*11156 (-25.23%)*	**14199**	15662 (10.30%)	11/12
	3	24	3	72	*32124 (19.93%)*	**16643**	35380 (112.58%)	6/12
	6	24	10	174	*41630 (9.99%)*	**18765**	35940 (91.53%)	4/12
	12	24	10	357	47577 (-2.17%)	**21810**	40311 (84.83%)	9/12
h2o: large**	1	36	300	21	*10570 (-29.15%)*	14199	**14030 (-1.19%)**	6/15
	3	24	100	103	*33592 (25.41%)*	**16643**	44369 (166.59%)	12/15
	6	24	10,10	171	*45429 (20.03%)*	**18765**	38225 (103.70%)	1/15
	12	24	300	497	*49369 (1.52%)*	**21810**	43456 (99.24%)	15/15
*h2o: large****	*1*	*24*	*300*	*19*	*17245 (-6.86%)*	*16419*	***12829 (-21.87%)***	*9/15*
	3	*24*	*100*	*53*	*23852 (-10.95%)*	*16643*	***13404 (-19.46%)***	*9/15*
	6	*24*	*300*	*118*	*33560 (-11.33%)*	*18765*	***16471 (-12.22%)***	*9/15*
	12	*24*	*300*	*234*	*41341 (-14.99%)*	***21810***	*22790 (4.49%)*	*8/15*

* The models were developed by multi-neural network strategy.

** The models were developed by iterative strategy.

*** These large multi-neural network models are given for comparison, as these had great performance on both validation and test set. The rank in last column shows the rank when comparing validation set results.